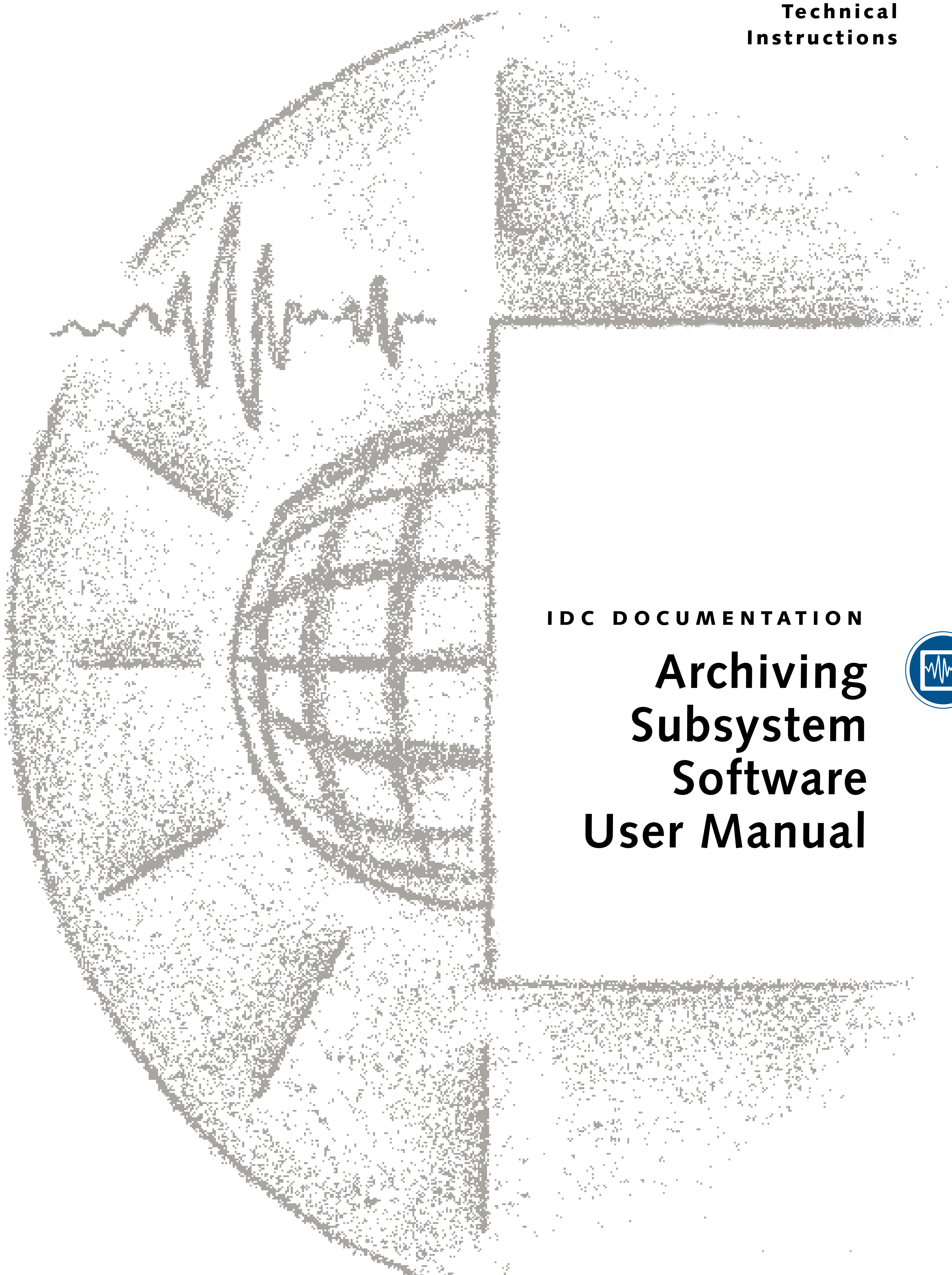


Technical
Instructions

IDC DOCUMENTATION

Archiving Subsystem Software User Manual

Notice
This document was published September 2001 by the Monitoring Systems Operation of Science Applications International Corporation (SAIC) as part of the International Data Centre (IDC) Documentation. Every effort was made to ensure that the information in this document was accurate at the time of publication. However, information is subject to change.

Contributors
David H. Salzberg, Science Applications International Corporation
Michael W. Zatloukal, Science Applications International Corporation

Ordering Information
The ordering number for this document is SAIC-01/3026.

This document is cited within other IDC documents as [IDC6.5.22].

Archiving Subsystem Software User Manual

CONTENTS

Archiving Subsystem Software User Manual

FIGURES

Archiving Subsystem Software User Manual

TABLES

About this Document

This chapter describes the organization and content of the document and includes the following topics:

- Purpose
- Scope
- Audience
- Related Information
- Using this Document

About this Document

PURPOSE

This document describes how to use the Archiving Subsystem software of the International Data Centre (IDC). The software is a computer software component (CSC) of the Data Management computer software configuration item (CSCI).

SCOPE

The manual includes instructions for setting up the software, using its features, and basic troubleshooting. This document does not describe the software's design or requirements. These topics are described in sources cited in "Related Information."

AUDIENCE

This document is intended for the first-time or occasional user of the software. However, more experienced users may find certain sections useful as a reference.

RELATED INFORMATION

The following document complements this document:

- *Archiving Subsystem* [IDC7.5.1]

See "References" on page 65 for a list of documents that supplement this document. The following UNIX manual (man) pages apply to the existing Archiving Subsystem software:

- *Archive*
- *MSwriter*

USING THIS DOCUMENT

This document is part of the overall documentation architecture for the IDC. It is part of the Technical Instructions category, which provides guidance for installing, operating, and maintaining the IDC systems. This document is organized as follows:

- Chapter 1: Introduction

  This chapter provides an overview of the software's capabilities, development, and operating environment.

- Chapter 2: Operational Procedures

  This chapter describes how to use the software and includes detailed procedures for startup and shutdown, basic and advanced features, security, and maintenance.

- Chapter 3: Troubleshooting

  This chapter describes how to identify and correct common problems related to the software.

- Chapter 4: Installation Procedures

  This chapter describes first how to prepare for installing the software, then how to install the executable files, configuration data files, database elements, and any Tuxedo files. It also describes how to initiate operation and how to validate the installation.

- References

  This section lists the sources cited in this document.

- Glossary

  This section defines the terms, abbreviations, and acronyms used in this document.

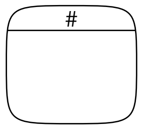
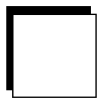
- Index

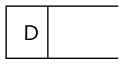
  This section lists topics and features provided in this document along with page numbers for reference.

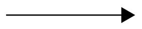
Conventions

This document uses a variety of conventions, which are described in the following tables. Table I shows the conventions for data flow diagrams. Table II lists typographical conventions.

TABLE I: DATA FLOW SYMBOLS

Description	Symbol[1]
process	
external source or sink of data	

data store
M = memory store
D = disk store
Db = database store
MS = mass store
	

data flow	

1. Most symbols in this table are based on Gane-Sarson conventions [Gan79].

TABLE II: TYPOGRAPHICAL CONVENTIONS

Element	Font	Example
database table	**bold**	**dataready**
database table and attribute, when written in the dot notation		**prodtrack.***status*
database attributes	*italics*	*status*
processes, software units, and libraries		*ParseSubs*
user-defined arguments and variables used in parameter (par) files or program command lines		`LOGDIR=`*/logs*
titles of documents		*Archiving Subsystem*
computer code and output	`courier`	`par=$(IMSPAR)`
filenames, directories, and websites		`Archive.par`
text that should be typed exactly as shown		`ps -ef | grep MSwriter`

Chapter 1: Introduction

This chapter provides a general description of the software and includes the following topics:

- Software Overview
- Status of Development
- Functionality
- Inventory
- Environment and States of Operation

Chapter 1: Introduction

SOFTWARE OVERVIEW

The software of the IDC acquires time-series and radionuclide data from stations of the International Monitoring System (IMS) and other locations. These data are passed through a number of automatic and interactive analysis stages, which culminate in the estimation of location and origin time of events (earthquakes, volcanic eruptions, and so on) in the earth, including its oceans and atmosphere. The results of the analysis are distributed to States Parties and other users by various means. Approximately one million lines of developmental software are spread across six CSCIs of the software architecture. One additional CSCI is devoted to run-time data of the software. Figure 1 shows the logical organization of the IDC software. The Data Management CSCI receives and archives data through the following CSCs:

- Data Archiving

  This software migrates data from the operations database to the archive database. It also archives waveform data on the mass-storage device and fills the archive of waveform segments associated to events.

- Database Libraries

  This software consists of libraries and include files needed for database access.

- Database Tools

  This software consists of utilities used in connection with database access.

- Configuration Management

  This software consists of scripts that facilitate recurring tasks in configuration management.

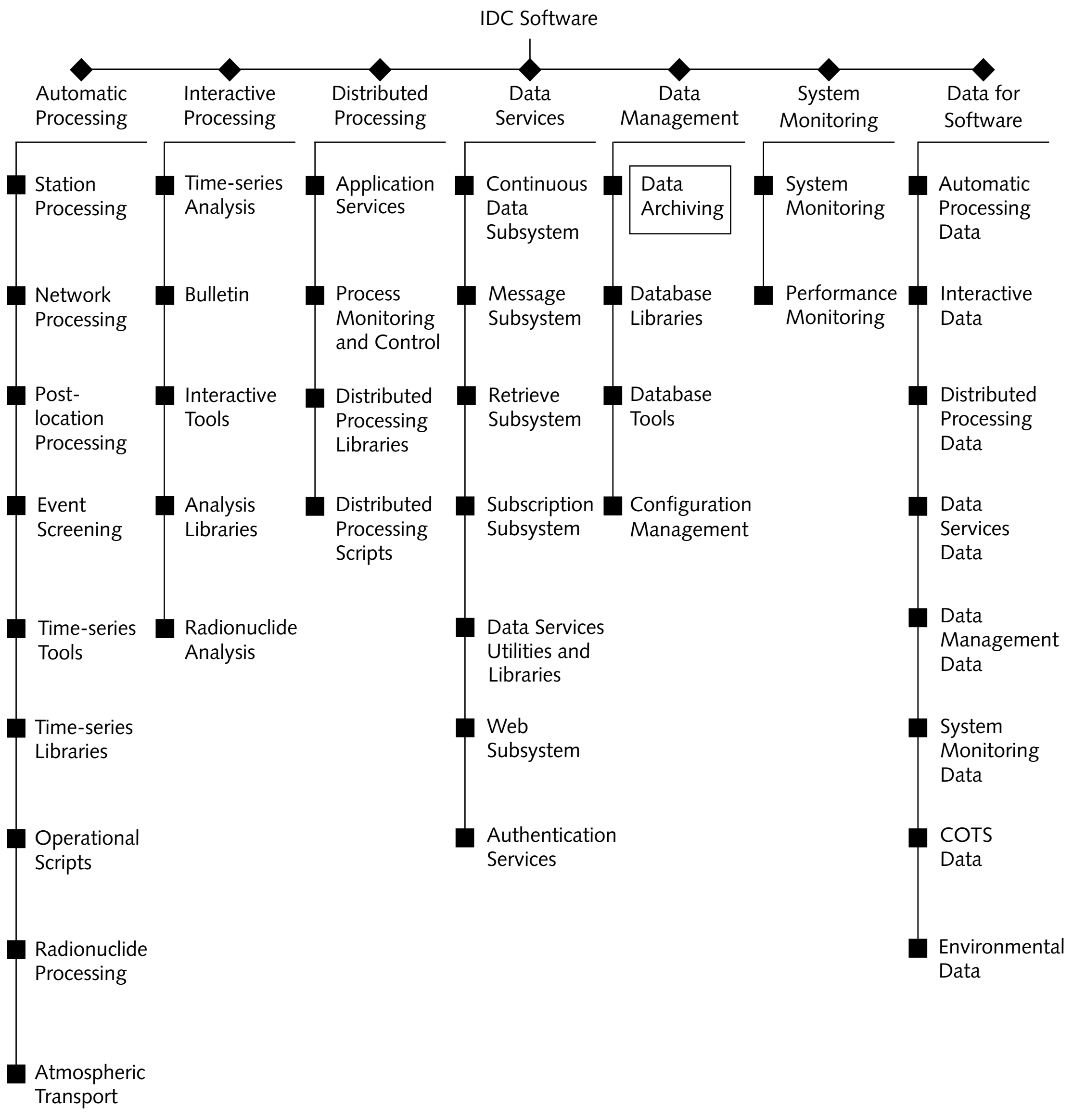


FIGURE 1. IDC SOFTWARE CONFIGURATION HIERARCHY

Figure 1 shows the logical organization of the IDC software. The Archiving Subsystem is part of the Data Archiving CSC, which is one component of the Data Management CSCI.

Figure 2 shows an example (using primary and auxiliary waveforms) of the role of the Archiving Subsystem, which is to archive items stored as files on the UNIX filesystem and referenced by database records that include file locations and data size. These database records may include supplementary information such as station information or time. Figure 2 also shows waveform data, messages, and file references created by the Continuous Data Subsystem and Message Subsystem.

The Segment Archive archives data on the mass-storage device and fills the archive of waveform segments associated to events. This Software User Manual does not address the Segment Archive.

STATUS OF DEVELOPMENT

The Archiving Subsystem is complete.

FUNCTIONALITY

The primary function of the Archiving Subsystem is to facilitate permanent storage of data in a manner ensuring complete data archiving while allowing for data retrieval at a later time. Data files (waveforms, messages, and file products) are transferred directly to the mass-storage device. The associated database tables (**wfdisc**, **msgdisc**, and **fileproduct**) are transferred from the operations database to the archive database. Some attributes of the database tables (directory path, file name, and file offset) are modified to reflect the location of the data on the archive filesystem. *Archive* defines and queues the intervals to be archived, reads the database records referencing the data, reads the data files, transmits data files to *MSwriter*, and writes to the archive database. *MSwriter* is a light-weight server run on the computer with the mass-storage device (mass-storage server). It accepts the data, writes to the mass-storage device, and reports errors to the client software (*Archive*). The Archiving Subsystem runs independent of the Distributed Application Control System (DACS) and can be initiated by *cron*. *Archive* is monitored using the *WorkFlow* graphical user interface (GUI).

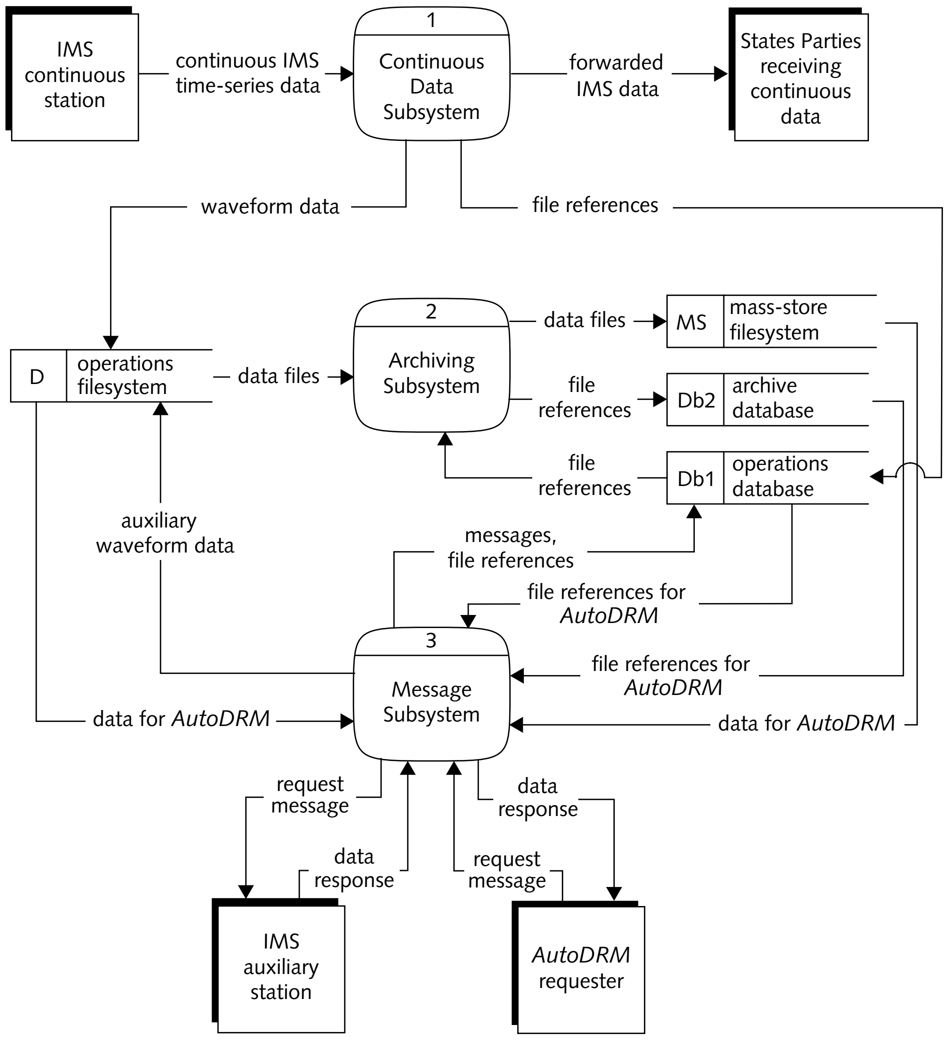


FIGURE 2. RELATIONSHIP OF ARCHIVING SUBSYSTEM TO CONTINUOUS DATA AND MESSAGE SUBSYSTEMS

Features and Capabilities

The Archiving Subsystem manages the transfer of data and information from operational filesystems to an archival storage area. The data types configured in the released software for archiving are waveform data, IMS messages, and station capability reports. Other data types, such as the CD-1.1 Frame Store, Radionuclide Data, and Threshold Monitoring Results, can be configured using the file product interface, which allows any filesystem object to be represented by *dir*, *dfile*, *foff*, and *size* in a database table (**fileproduct**).

The Archiving Subsystem uses a transactional client/server design. The *Archive* client software reads the database records and corresponding data files from the operations database and filesystem. The data are transferred to *MSwriter* (the server software), which writes to the archive filesystem (typically a type of mass-storage device) and notifies *Archive* of any errors. *Archive* then updates the data references to reflect the archived file location and inserts the database records in the archive database.

The Archiving Subsystem organizes the data into intervals during the archiving process to facilitate easy retrieval, queueing, and tracking. The Archiving Subsystem also monitors the status of archiving, as failure to archive data can result in permanent data loss.

Performance Characteristics

The performance of the Archiving Subsystem is limited by the available network bandwidth and read/write speed. Errors with the subsystem are infrequent and usually the result of a system error (availability of the mass store, network connections, and so forth). Archiving Subsystem testing has demonstrated its accuracy and reliability to be greater that 99.99 percent.

Archive performance is primarily affected by network bandwidth. Thus, the primary waveform archive should take the most time to complete. Observationally, the typical archiving time per day across a 10 Megabits/second LAN for the various archived data types are listed in Table 1.

TABLE 1: TYPICAL PERFORMANCE OF THE ARCHIVING SUBSYSTEM

Data type	MB per Day	Average Run Time per Day
Waveforms	3506	6 hr
IMS Messages	120	30 min
Station and Channel Capability Reports	1	2 min

Related Tools

The *WorkFlow* graphical user interface (GUI) tool is used to monitor the performance of the Archiving Subsystem. *WorkFlow* configured for archive monitoring displays the status of intervals being archived. The script *check_archive* is used to verify the proper functioning of the Archiving Subsystem. Procedures for using *WorkFlow* are presented in "Chapter 3: Troubleshooting" on page 31. Procedures for using *check_archive* are discussed in "Validating Installation" on page 61.

INVENTORY

The Archiving Subsystem is composed of executables, libraries, database tables and accounts, and parameter files shown in Table 2. Table 2 shows not only Archiving Subsystem components, but also components common to the Archiving and other subsystems.

TABLE 2: ARCHIVING SUBSYSTEM INVENTORY

Item	Type
Archive	executable
check_archive	executable
MSwriter	executable
WorkFlow	executable (Distributed Processing CSCI)
`Archive.par`	parameter file

TABLE 2: ARCHIVING SUBSYSTEM INVENTORY (CONTINUED)

Item	Type
`MSwriter.par`	parameter file
`WorkFlow_Arch.par`	parameter file
liblogout	common library
libpar	common library
libinterp	common library
libstdtime	common library
libgdi	common library
libsocket	COTS library
libnsl	COTS library
libm	COTS library
libdl	COTS library
arch_data_type	database table (operations read)
chan_groups	database table (operations read)
dlfile	database table (operations read)
fileproduct	database table (operations read, archive write)
fpdescription	database table (operations read)
interval	database table (operations read/write)
lastid	database table (operations read/write)
lastid_archdb	database table (archive read/write)
msgdisc	database table (operations read, archive write)
wfaux	database table (operations read, archive write)
wfdisc	database table (operations read, archive write)

ENVIRONMENT AND STATES OF OPERATION

The following sections describe the commercial-off-the-shelf (COTS) software and hardware required to operate the Archiving Subsystem.

Software Environment

The Archiving Subsystem software requires a relational database both for its internal processing and as a source and destination for migration of the associated database tables. The software was tested using ORACLE 8i and a Solaris 7 UNIX operating system.

Hardware Environment

The Archiving Subsystem does not require significant computer resources (32 MB RAM, disk space for log files and run-time parameters) and runs locally on the computer with the data that are to be archived and on the mass-storage server. The host (where *Archive* is run) is the computer with the data for the particular data type. The mass-storage server (where *MSwriter* is run) is the computer where the mass-storage device resides. For example, the host for continuous waveform archiving is the disk loop host (DAQ1HOST), which is a Sun Enterprise Server with approximately 115 GB of disk space (this value increases when new stations are brought on-line). The storage requirements for the DAQ1HOST are determined by the storage needs of the disk-loop data files and are independent of the Archiving Subsystem. Similarly, the processing load is driven by signal processing of the disk-loop data and not by the Archiving Subsystem. The server for continuous waveform archiving has disk space for its own operation as well as the storage associated with the mass-storage device. *MSwriter* requires only a modest amount of disk space on the server, whereas the storage requirements for the mass-storage device are determined by the data accumulation rate. Figure 3 shows a representative hardware configuration.

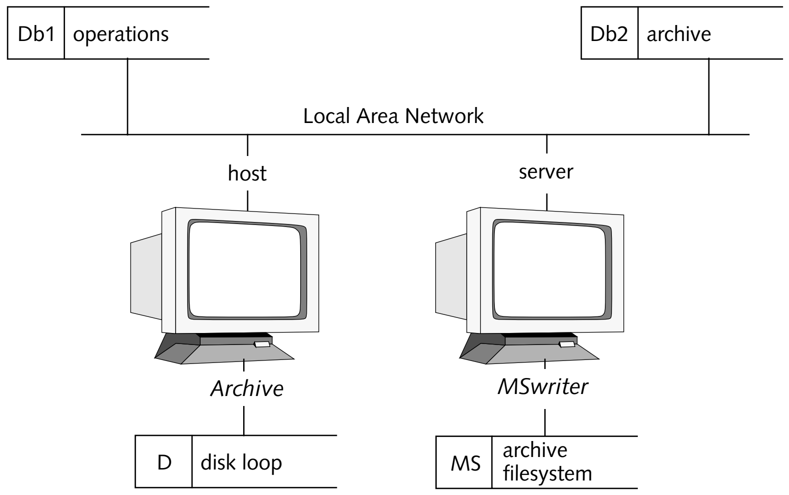


FIGURE 3. ARCHIVING SUBSYSTEM HARDWARE CONFIGURATION

Normal Operational State

The Archiving Subsystem normally runs from *cron* on a set schedule. The Archiving Subsystem has four `crontab` entries for archiving messages, station capability reports, primary waveform data, and auxiliary waveform data. Details on these entries and initial setup are provided in "Chapter 4: Installation Procedures" on page 43.

Contingencies/Alternate States of Operation

You have the option of executing the subsystem from a command line interface if desired. If the data center is experiencing a backlog of data to archive, or if the Archiving Subsystem encounters significant errors where intervals are not progressing to *state* = `DONE`, you may run the subsystem independently from *cron* to "catch up." Copy or type the commands from the `crontab` files to execute *Archive* for different data types. For example:

```
/cmss/rel/bin/Archive Datatype=PRIARC \
Par=/cmss/config/app_config/archive/Archive.par
```

You may also edit `crontab` files so that *cron* runs *Archive* more frequently. For example:

```
#
#----------Run Archiver for Continuous Waveforms
#
03 18,20,22,0,2,4  * * * ( env ‘cat /cmss/config/
system_specs/env/global.shenv‘ run_archive PRIARC >/
dev/null 2>&1 )
```

If a critical failure of the mass-storage device hardware occurs, you can change the archive data destination to an alternate mass store. The mass store location is set with a parameter in the file `shared.par`. Modify this setting to make use of other available storage in an emergency. Then install *MSwriter* on the new archiving destination using procedures provided in "Chapter 4: Installation Procedures" on page 43.

Chapter 2: Operational Procedures

This chapter provides instructions for using the software and includes the following topics:

- Software Startup
- Software Shutdown
- Basic Procedures
- Advanced Procedures
- Maintenance
- Security

Chapter 2: Operational Procedures

SOFTWARE STARTUP

The Archiving Subsystem is designed to run automatically without user assistance. Entries in the `crontab` file are made during installation to schedule automatic execution of the subsystem. The *cron* daemon invokes *Archive*, which invokes (via the *inetd* daemon) *MSwriter* by connecting to the correct port (specified in `Archive.par` by [*datatype*]-*Archive_Port* and *archive_machine*).

SOFTWARE SHUTDOWN

The Archiving Subsystem can be shut down by disabling *Archive* `crontab` entries and killing both *Archive* and *MSwriter* processes. To disable *Archive* `crontab` entries, log onto the machine with the particular data type to be archived and comment out its entry from the `crontab` file by editing the file with the `crontab -e` command. Insert a number sign (#) before the appropriate row and exit the editor. After the `crontab` entries are disabled, kill the *Archive* process with the UNIX `kill` command:

```
% ps -ef | grep Archive
cmss 456 1 0 Jun 20 pts/8 1:67 Archive par=/cmss/config/app_config/
archive/Archive.par
% kill 456
```

Log onto the mass-storage server and search for any active *MSwriter* processes using the UNIX `ps` command. Stop any *MSwriter* processes with the UNIX `kill` command:

```
% ps -ef | grep MSwriter
cmss 19799 1 0 Jun 20 pts/8 1:41 MSwriter par=/cmss/config/
app_config/archive/MSwriter_sock.par
% kill 19799
```

BASIC PROCEDURES

The basic procedures for the Archiving Subsystem are to check its status and to purge operational data after the data are archived. These topics are addressed in "Monitoring" on page 32 and "Maintenance" on page 27, respectively.

Obtaining Help

UNIX man pages are available for the Archiving Subsystem software components *Archive* and *MSwriter*. For information on diagnosing Archiving Subsystem problems see "Chapter 3: Troubleshooting" on page 31.

ADVANCED PROCEDURES

Several advanced Archiving Subsystem procedures are available. The sections "Understanding the Software Configuration Model" and "Delivered Configuration" provide critical background information about the configuration of *Archive*. Advanced procedures include modifying *Archive* parameters, creating and adding new data types, and creating and adding new fileproducts. These procedures are explained in the sections that follow.

Understanding the Software Configuration Model

The Archiving Subsystem software is configured from two sources: database tables and the `Archive.par` file.

The database tables **arch_data_type** and **chan_groups** are used to store configuration information that is used to link data across database tables. For example, the *sta* and *chan* attributes in **chan_groups** are linked to the *sta* and *chan* attributes in **wfdisc**; and the *class* and *name* attributes in **chan_groups** are linked to the *class* and *name* attributes in **interval**.

The `Archive.par` file contains two sections of parameter settings. The first section contains configuration settings (for example, archiving schedule, location of the mass-storage device, and so forth). The second section contains configuration rules in the form of SQL*Plus queries for each data type. An example `Archive.par` is shown in "Archive.par" on page 46.

Delivered Configuration

The `Archive.par` file is released with preconfigured SQL statements that provide the interval creation and queuing rules. Table 3 lists the data types and the preconfigured rules for creating and queuing intervals for each type.

WARNING: Modifying the SQL rules can cause *Archive* to fail. Refer to " Configuration Data Files" on page 45 for more information.

TABLE 3: DELIVERED CONFIGURATION FOR DATA TYPES

Data Types	Interval Creation Basis	Interval Queuing Basis
`waveforms: Primary`	**dlfile** entries exist	3 days
`waveforms: Auxiliary`	**wfdisc** entries exist	8 days
`file products: STACAP`	file products exist	20 days
`Messages`	messages exist	20 days

Another aspect of the configuration is the organization of archived files. The subsystem creates one archived file for each processing interval. The rules for interval creation are straightforward for messages, framestores, and station capability reports; however, finer control of both primary and auxiliary waveform file content is possible by grouping stations/channels. This fine control is achieved by assigning several *sta* and *chan* rows the same *name* and *class* in the **chan_groups** table. The **interval** table contains the *name* and *class* of the data to archive, so the stations/ channels grouped into the same *name* and *class* are archived as a single file. As delivered, all auxiliary waveforms are grouped together (*class* = `AUXARC`, *name* = `AUXNET`) and all primary waveforms from a given array are grouped together (*class* = `PRIARC`, *name* = *array_code*).

Modifying Archive Parameters

The archiving process is controlled through parameterized SQL statements stored in `Archive.par`. Many of the table names and values in the SQL statements are redirected values (that is, parameter values set in another par file). The following parameters in the `Archive.par` file can be modified with careful planning:

Operations database account name:

```
account=$(ops_acct)
```

Table names:

```
MSGDISC=$(account).msgdisc
WFDISC=$(account).wfdisc
DLFILE=$(account).dlfile
FILEPRODUCT=$(account).fileproduct
FPDESCRIPTION=$(account).fpdescription
INTERVAL=interval
CHANGROUPS=$(account).chan_groups
```

Values of the **interval**.*state* (where [*datatype*] is the data type):

```
[datatype]-interval-done=DONE
[datatype]-interval-failed=FAILED
[datatype]-interval-new=NEW
[datatype]-interval-nodata=NODATA
[datatype]-interval-queued=QUEUED
[datatype]-interval-running=RUNNING
```

Maximum size of archived files[1]:

```
[datatype]-maxfilesize=2147483647
```

Generic parameters[2]:

```
[datatype]-class=MSG
[datatype]-archive_directory=$(ARCHIVE_MSG_DIR)
[datatype]-Archive_Port=3888
[datatype]-fileprefix=Arch
[datatype]-filepost='.w'
[datatype]-lastid_table='lastid_arcdb';
[datatype]-lastid_keyname=wfid
[datatype]-archive-days=20
[datatype]-lookback-days=10
```

You can reset how far back *Archive* scans by updating the value of [*datatype*]-*look-back-days*. You can change the latency of primary waveform archiving by updating [*datatype*]-*archive-days* (for example, five days instead of three days).

Occasionally you may need to modify the SQL rules to, for example, fix errors or change the criteria for creating new archiving intervals for a certain data type.

Another aspect of the configuration that you may change is the grouping of the stations/channels into files. Grouping is handled in the **chan_groups** database table. The **chan_groups** table relates *class* and *name* in the **interval** to *sta* and *chan* in the **wfdisc** (or other) tables. As delivered, *Archive* combines all elements in an array into a single interval. For example, the elements of the PDAR array map as follows:

```
SQL> select distinct class, name, sta, chan from idcx.chan_groups
2  where name='PDAR';

CLASS            NAME             STA    CHAN
---------------- ---------------- ------ --------
PRIARC           PDAR             PD01   sz
PRIARC           PDAR             PD02   sz
PRIARC           PDAR             PD03   sz
```

1. The maximum file size must not exceed the UNIX file size limit.
2. If the port number is changed, the `inetd.conf` file must also be changed.

```
PRIARC          PDAR            PD04   sz
PRIARC          PDAR            PD05   sz
PRIARC          PDAR            PD06   sz
PRIARC          PDAR            PD07   sz
PRIARC          PDAR            PD08   sz
PRIARC          PDAR            PD09   sz
PRIARC          PDAR            PD10   sz
PRIARC          PDAR            PD11   sz
PRIARC          PDAR            PD12   sz
PRIARC          PDAR            PD13   sz
PRIARC          PDAR            PD31   be
PRIARC          PDAR            PD31   bn
PRIARC          PDAR            PD31   bz
PRIARC          PDAR            PD32   se
PRIARC          PDAR            PD32   sn
PRIARC          PDAR            PD32   sz
```

To change the grouping so all primary stations are written into a single file, you would update the **chan_groups** table assigning the same value to the *name* attribute for all elements with *class* = `PRIARC`. Auxiliary waveforms are assigned *class* = `AUXARC`, *name* = `AUXNET`.

Adding New Stations

To add a new station to the system, a row must be added to the **chan_groups** table for each channel of data. The **chan_groups** table includes *class*, *name*, *sta*, *chan*, and other attributes. The *class* attribute groups stations into collections of similar data types; primary seismic stations are all in *class* = `PRIARC` and auxiliary seismic stations are all in *class* = `AUXARC`. The *name* attribute is the name (station code) that is given to the entire array or 3-C station for primary seismic stations; all auxiliary seismic stations are given *name* = `AUXNET`. If the station is an array, then each of the array elements is given a different *sta* value; if the station is 3-C, then the *sta* attribute is the same as the *name* attribute.

To add a new 3-C station to the archiving process, new rows must be added to the **chan_groups** table for each new channel of data. For example, to add the primary 3-C seismic station BOSA, its *class*, *name*, and *sta* would be `PRIARC`, `BOSA`, and `BOSA`, respectively. The following example SQL statements insert rows into the **chan_groups** table for three channels (bz, bn, and be) of the BOSA station:

```
insert into chan_groups
    values ('PRIARC','BOSA','BOSA','bz',-1,-1,
    to_number(to_char(sysdate,'yyyyddd'),'9999999'),-1,sysdate);

insert into chan_groups
    values ('PRIARC','BOSA','BOSA','bn',-1,-1,
    to_number(to_char(sysdate,'yyyyddd'),'9999999'),-1,sysdate);

insert into chan_groups
    values ('PRIARC','BOSA','BOSA','be',-1,-1,
    to_number(to_char(sysdate,'yyyyddd'),'9999999'),-1,sysdate);
```

To add a new station to an existing array, find the *class* for that array. For example, to add channel bz to element YKW3 of array YKA, first find the *class* for YKA as follows:

```
SQL> select class from idcx.chan_groups where name='YKA';

CLASS
--------
PRIARC
PRIARC
...
```

Next, add the new row to the **chan_groups** table with the proper *class*, *name*, *sta*, and *chan* attributes using the following SQL statement:

```
insert into chan_groups
    values('PRIARC','YKA','YKW3', 'bz', -1, -1,
    to_number(to_char(sysdate,'yyyyddd'),'9999999'),-1,sysdate);
```

To add an entire array to the archiving process, new rows must be added to the **chan_groups** table for each new channel of data. For example, to add the primary seismic array UKR, its *class* and *name* would be PRIARC and UKR. The following example SQL statements insert rows into the **chan_groups** table for three elements (UKR01, UKR02, and UKR03) of the UKR array, assuming one channel per element:

```
insert into chan_groups
    values ('PRIARC','UKR','UKR01','SHZ',-1,-1,
    to_number(to_char(sysdate,'yyyyddd'),'9999999'),-1,sysdate);
insert into chan_groups
    values ('PRIARC','UKR','UKR02','SHZ',-1,-1,
    to_number(to_char(sysdate,'yyyyddd'),'9999999'),-1,sysdate);
insert into chan_groups
    values ('PRIARC','UKR','UKR03','SHZ',-1,-1,
    to_number(to_char(sysdate,'yyyyddd'),'9999999'),-1,sysdate);
```

Creating and Adding New Data Types

Data are archived based on their “data type.” For example, the rules for archiving primary and auxiliary waveform data are different because they are collected and processed differently. The data types for primary and auxiliary data are `PRIARC` and `AUXWF`, respectively.

You must perform several steps to add a new data type to the Archiving Subsystem. These steps include 1) adding a new row to the **arch_data_type** table; 2) defining SQL*Plus rules for creating, queueing, and reading intervals; and 3) adding new parameters to `Archive.par`.

The **arch_data_type** table contains configuration information such as the data type, the table name to archive, the primary and secondary keys to the table, and whether or not adjacent records should be merged (merging applies only to **wfdisc** records).

Use the following procedure to add a new data type to the Archiving Subsystem:

1. Insert a row in the **arch_data_type** table using an SQL*Plus command similar to the following example for the **wfdisc** table:

   ```
   SQL> insert into arch_data_type values
    2   (9,'PRIARC','wfdisc','sta','time','y','n',-1,-1,sysdate);
   ```

2. Next, develop archiving rules for the new data type. Rules are needed to create intervals, queue intervals, read queued intervals, and read data for an interval. Archiving rules are usually developed through experimentation in an SQL*Plus session.

   To develop a rule that creates intervals, select the information needed to insert into the **interval** table for some time span; then subtract the information that is already in the **interval** table for that time span. As an example for primary seismic waveform data, the following query selects all of the disk loop information for a five-day interval and then removes the information that already exists in the **interval** table for that same time span:

```
select distinct -1 intvlid, 'PRIARC' class, cg.name name,
    dl.time time, dl.time+dl.tlen endtime, 'NEW' state,
    sysdate moddate, sysdate lddate
from idcx.dlfile dl, idcx.chan_groups cg
where cg.sta=dl.sta
  and cg.chan=dl.chan
  and dl.time > 993427200.00 - 86400*5
  and cg.class='PRIARC'
minus
select distinct -1 intvlid,'PRIARC' class, cg.name name,
    i.time time, i.endtime, 'NEW' state, sysdate moddate,
   sysdate lddate
from idcx.chan_groups cg, idcx.interval i
where cg.name=i.name
  and cg.class=i.class
  and i.class='PRIARC'
  and i.time > 993427200 - 86400*5;
```

   When you are satisfied with your rule, then place the rule in the `Archive.par` file and substitute parameters for the class and state values, the current dates and times, the table names, and the number of lookback days. The following example is a parameterized version of the previous rule placed in the *PRIARC-interval_create* parameter:

```
PRIARC-interval_create="\
select distinct -1 intvlid, '$(PRIARC-class)' class, \
    cg.name name, dl.time time, dl.time+dl.tlen endtime, \
    '$(state-NEW)' state, \
    to_date('{current_lddate}','YYYYMMDD HH24:MI:SS') moddate, \
    to_date('{current_lddate}','YYYYMMDD HH24:MI:SS') lddate \
from $(DLFILE) dl, $(CHANGROUPS) cg \
where cg.sta=dl.sta \
  and cg.chan=dl.chan \
  and dl.time > {current_epoch}-86400*$(PRIARC-lookback-days) \
  and cg.class='$(PRIARC-class)' \
minus \
select distinct -1 intvlid, '$(PRIARC-class)' class, \
    cg.name name, i.time time, i.endtime, \
    '$(state-NEW)' state, \
    to_date('{current_lddate}','YYYYMMDD HH24:MI:SS') moddate, \
    to_date('{current_lddate}','YYYYMMDD HH24:MI:SS') lddate \
from $(CHANGROUPS) cg, $(INTERVAL) i \
where cg.name=i.name \
  and cg.class='$(PRIARC-class)' \
  and i.class='$(PRIARC-class)' \
  and i.time > {current_epoch} - 86400*$(PRIARC-lookback-days)"
```

The values in {} are replaced at execution; `{current_epoch}` gives the current epoch time and `{current_lddate}` gives the current load date.

3. Add rules to update the **interval** table, select intervals with *state* = `queued`, and read the data. These parameterized rules for the example follow:

```
PRIARC-interval_update="\
update $(INTERVAL) \
set state='$(state-QUEUED)', \
moddate=to_date ('{current_lddate}','YYYYMMDD HH24:MI:SS') \
where class='$(PRIARC-class)' \
  and state in ('$(state-NEW)', '$(state-RETRY)') \
  and time < {current_epoch} - $(PRIARC-archive-days) *86400 \
  and class='$(PRIARC-class)'"

PRIARC-interval_select="\
select * from $(INTERVAL) \
where state='$(state-QUEUED)' and class='$(PRIARC-class)' \
order by time"

PRIARC-read_data="\
select distinct w.* from $(WFDISC) w, $(CHANGROUPS) cg \
where cg.sta=w.sta and cg.chan=w.chan \
  and w.time between %f and %f-.00001 and cg.name='%s' \
  and cg.class='%s' and cg.offdate=-1 \
order by w.sta, w.chan, w.time"
```

In the read data query, the two `%f` symbols give the start and end time of the intervals and the two `%s` symbols refer to the class and name. The `%s` and `%f` are used by the 'C' function `sprintf` for formatting output. Refer to the `sprintf` UNIX man page for more details.

After updating the par file (including adding the data type-specific parameters), test the values using the runtime flag *debug_queries*, which prints (to the log file) the database queries *Archive* executes to update and read intervals (*Archive* then exits). Also test *Archive* prior to installing the new configuration into operations.

Creating and Archiving New File Products

The Archiving Subsystem supports user-defined file products. The software design that underlies this capability is illustrated in the following procedure (example included), which is used to define a file product.

To define a file product, add a record describing the file product to the **fpdescription** database table (see [IDC5.1.1Rev2]). You must provide values for the unique identifier (*typeid*), the product type, the description of the product, the type of data, and the format of the data.

After the description is created, insert a **fileproduct** row referencing the filesystem object. The **fileproduct** row contains the directory *(dir)*, filename *(dfile)*, file offset *(foff)*, and product size *(dsize*), as required by the Archiving Subsystem. It also contains the *typeid* (which links to the **fpdescription** table), the time and end time of the object, and the station and channel codes if appropriate for the file object. Add **fileproduct** rows to the database either by inserting the record manually, by using the Data Services CSCI library *libfileproduct*, or by using the Subscription Subsystem application *write_fp* [IDC7.4.4]. Refer to the *write_fp* man page.

After defining the file product, the Archiving Subsystem must be configured to handle the new product. To configure the Archiving Subsystem:

1. Add a row to the **arch_data_type** table for the new data type.
2. Define the data type-specific parameters to create, queue, and read processing intervals from the **interval** table.
3. Define the data type-specific parameters to read the **fileproduct** database rows.

As an example of this procedure, the following steps add station capability reports to the archiving process:

1. Add a row to the **arch_data_type** table using the following SQL statement:

   ```
   insert into arch_data_type
   values (3,'STACAP','fileproduct','sta','time',-1,-1,sysdate);
   ```

2. Define the data type-specific configuration rules. For *datatype* `STACAP`, the interval creation rules are to generate the **interval** rows for the reports on a daily basis for the previous 30 days. The following parameter is an SQL*Plus statement that creates the internal rows:

   ```
   STACAP-interval_create="select distinct -1 intvlid, \
       'CAPARC' class, 'STACAP' name, \
       floor(fp.time/86400)*86400 time,\
       floor((fp.time+86400)/86400)*86400 endtime, 'NEW' state,\
       to_date('{current_lddate}', 'YYYYMMDD HH24:MI:SS') moddate,\
       to_date('{current_lddate}','YYYYMMDD HH24:MI:SS') lddate \
   from fileproduct fp \
   where fp.typeid in \
       (select typeid from fpdescription \
        where prodtype in ('STA_STATUS', 'CHAN_STATUS')) \
     and fp.time > {current_epoch} - 30 *86400 \
   minus \
   select distinct -1 intvlid, 'CAPARC' class, 'STACAP' name,\
       i.time, i.endtime, 'NEW' state, \
       to_date('{current_lddate }','YYYYMMDD HH24:MI:SS') moddate,\
       to_date('{current _lddate}','YYYYMMDD HH24:MI:SS') lddate\
   from $interval i \
   where i.time > {current_epoch} -86400*30 \
     and i.class='CAPARC' and i.name='STACAP'"
   ```

The *STACAP-interval_update* parameter defines the queuing rules. For STACAP the intervals are queued `$(stacap-archive-days)` days after the reports are generated.

```
STACAP-interval_update="update interval\
set state='QUEUED', \
    moddate=to_date('{current_lddate}', 'YYYYMMDD HH24:MI:SS') \
where class='CAPARC' \
  and state in ('NEW', 'RETRY') \
  and time < {current_epoch} - $(stacap-archive-days)*86400"
```

The *STACAP-interval_select* parameter reads all of the records with **interval**.*state* = `QUEUED` for *class* = `CAPARC`:

```
STACAP-interval_select="select * from $(INTERVAL) \
where state='QUEUED' \
and class='CAPARC' \
order by time"
```

The final parameterized database query is in the parameter *STACAP-read_data*. This query reads the **fileproduct** rows for the interval of interest. An example of the parameter value follows:

```
STACAP-read_data="select f.* from fileproduct f \
where f.typeid in
    (select typeid from fpdescription \
     where prodtype in ('STA_STATUS', 'CHAN_STATUS')) \
  and f.time between %f and %f-.00001 \
order by f.sta, f.time"
```

The first and second `%f`'s in this parameterized query are replaced by **interval**.*time* and **interval**.*endtime*, respectively. *Archive* performs this action using the C function `sprintf`.

3. To complete the configuration, set the following data type-specific parameters, which are discussed in the *Archive* man page:

```
STACAP-archive_directory
STACAP-lookback-days
STACAP-archive-days
STACAP-fileprefix
STACAP-filepost
```

MAINTENANCE

The primary maintenance task for the Archiving Subsystem is to purge files from operations after archiving. Continuous waveform data are automatically purged and do not require manual purging.

Purging Archived Files

Before purging, ensure that intervals to be archived have been successfully transferred to the mass store. The status of intervals on the *WorkFlow* GUI should all indicate "DONE" for all archived intervals for which the operational data are to be purged. All database references should also be present within the archive database. Lastly, all data files in the mass store should be identical to the data files in the operations filesystem. Refer to "Monitoring" on page 32 for additional instructions for verifying correct and complete archiving.

After determining that selected data have been properly transferred to the mass store, delete database references from the operations database and then delete the operations data files for the archived data. Examples are shown in the following paragraphs for deleting operations database records and associated files for both messages and auxiliary waveforms. Primary waveforms are purged by the Continuous Data Subsystem [IDC6.5.18] and do not need to be purged manually. Adapt the examples below for archiving any additional data types and file products created.

WARNING: Use extreme care while purging the operations data files and database records. If either the files or records are purged prior to archiving, their contents are lost.

Find the database records for the interval. The example given is for messages in an archived interval defined by *intvlid*. Create a file on the operations filesystem with the *dir/dfile* from the operations database. For example:

```
spool rm_ops_MSGS
select distinct 'rm '||m.dir||'/'m.dfile
from msgdisc m, interval i
```

```
where m.itime between i.time and i.endtime-.001
  and i.intvlid=[intvlid];
spool off
```

Use the following SQL command to delete the records from the operations database using the same clause as in the previous query:

```
delete from msgdisc
where msgid in
    (select msgid from msgdisc m, intvlid i
     where m.itime between i.time and i.endtime-.001
       and i.intvlid=[intvlid]);
...
commit;
quit;
```

A `commit` command is shown in the example for completeness. SQL will automatically commit the change when you quit the session. After the records are deleted from the operations database, run the script *rm_ops_MSGS.lst* created by the first query to remove the associated files:

```
% sh rm_ops_MSGS.lst
```

The following example purges operations files for auxiliary data (`AUXARC`). This example is based on the [*datatype*]-*read_data* parameter.

```
AUXWF-read_data="select distinct w.* \
from $(WFDISC) w, $(CHANGROUPS) cg \
where cg.sta=w.sta and cg.chan=w.chan \
  and w.time between %f and %f-.00001 \
  and cg.name='%s' and cg.class='%s' and cg.offdate=-1 \
order by w.sta, w.chan, w.time";
```

Create a file with the *dir*/*dfile*s on the operations filesystem:

```
spool rm_ops_AUX
select distinct 'rm '||w.dir||'/'w.dfile
from wfdisc w, chan_groups cg, interval i
where cg.sta=w.sta and cg.chan=w.chan and i.name=cg.name
```

```
  and i.class=cg.class and cg.offdate=-1 and
where w.time between i.time and i.endtime-%f-.00001 and
i.intvlid=[intvlid];
...
spool off
```

Delete the records from the operations database using the same clauses:

```
delete from wfdisc where wfid in
    (select distinct w.wfid
     from wfdisc w, chan_groups cg, interval i
     where cg.sta=w.sta and cg.chan=w.chan and i.name=cg.name
     and i.class=cg.class and cg.offdate=-1 and
where w.time between i.time and i.endtime -.00001 and
i.intvlid=[intvlid];);
...
commit;
quit;
```

After the operations database records have been deleted, run the script *rm_ops_AUX.lst* just created to remove the associated files:

```
% sh rm_ops_AUX.lst
```

SECURITY

Operators who have update permissions for database accounts can potentially remove, add or manipulate data in an account.

Passwords

Database passwords and accounts are stored in the file `process.par`. Access to this file is controlled by UNIX permissions and the operations manager controls UNIX group membership required to access `process.par`.

Chapter 3: Troubleshooting

This chapter describes how to identify and correct problems related to the Archiving Subsystem and includes the following topics:

- Monitoring
- Interpreting Error Messages
- Solving Common Problems
- Reporting Problems

Chapter 3: Troubleshooting

MONITORING

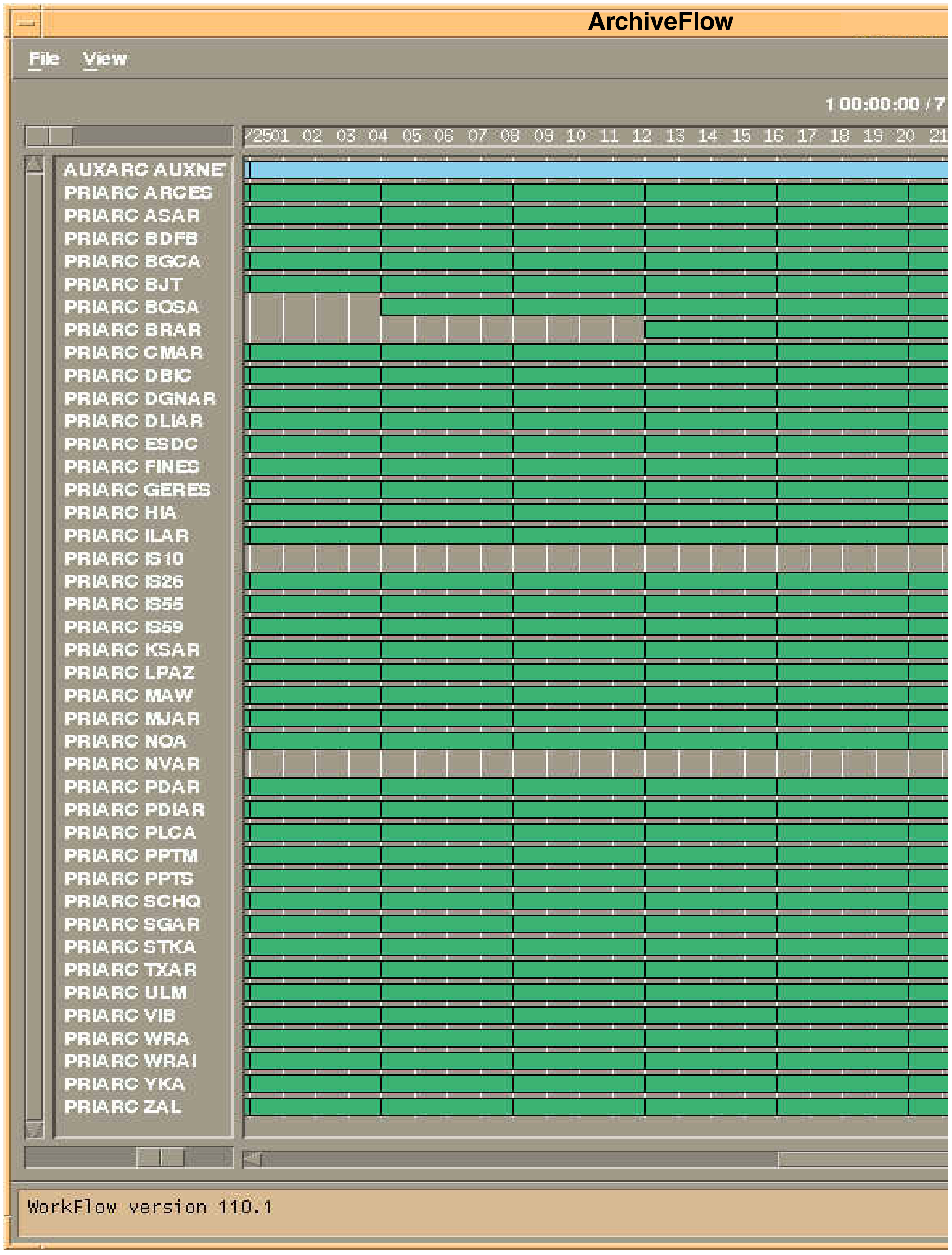
The primary method of monitoring Archiving Subsystem operation is through the use of the archiving version of the *WorkFlow* GUI tool. Other methods of monitoring the subsystem include tracking the interval processing in the **interval** database table using SQL*Plus commands, screening log files for problems, and running *check_archive*. These methods are described in the following sections.

Using WorkFlow for Archive

The recommended method for monitoring the Archiving Subsystem is through the use of the archiving version of the *WorkFlow* GUI. The *WorkFlow* program graphically displays the state of intervals in the **interval** table. Start the archiving version of the *WorkFlow* GUI by executing the *WorkFlow* binary with the `WorkFlow_arch.par` file. An example of a command line execution follows:

```
/home/cmss/rel/bin/WorkFlow \
par=/cmss/config/app_config/distributed/WorkFlow/WorkFlow_arch.par\
 -name Archiving WorkFlow
```

When *WorkFlow* is started from the command line, a new GUI window opens on your screen. Figure 4 is an example of the archive *WorkFlow* GUI window:



```
WorkFlow version 110.1
```

FIGURE 4. ARCHIVE WORKFLOW GUI

The left side of the *WorkfFlow* GUI displays (top to bottom) the auxiliary stations (AUXARC) and primary stations (PRIARC). Rows for station/capability reports (CAPARCH), and messages (MSG MSG) are listed last and are not shown in Figure 4. Intervals are depicted in different colors to represent the status of processing. Table 4 lists the archive *WorkFlow* GUI states and colors.

TABLE 4: WORKFLOW INTERVAL STATES AND COLORS

State	Color
NEW	blue
QUEUED	yellow
RUNNING	cyan
DONE	green
FAILED	red
RETRY	pink

The *WorkFlow* GUI legend window contains additional states that are not listed in Table 4. The script *check_archive* uses the VERIFIED and NONVERIF states internally during its processing. The states ARCNEW, ARCQUEUED, IGNORED, and PARCH5 are no longer used.

Checking Database Rows

Verify that the Archiving Subsystem has completely archived all intervals. As intervals are processed by the Archiving Subsystem, the interval's status in the **interval** table is progressively updated through the different states listed in Table 4.

Use SQL*Plus queries to ensure that intervals have been processed to a `DONE` *state*. After the Archiving Subsystem has processed all the intervals, log onto the database and select records from the **interval** table for the appropriate *class* as follows:

```
select state, count(*) from interval
where class='PRIARC"
group by state;
```

All intervals should have a `NEW` or `DONE` *state*. Any intervals with *state* = `QUEUED` indicates that *Archive* did not process the intervals. Any intervals with *state* = `FAILED` indicates that *Archive* was unable to process the intervals. Any intervals with *state* = `RUNNING` indicates that *Archive* is actively processing those intervals.

Any intervals with a `QUEUED`, `FAILED`, or `RUNNING` *state* at the completion of archiving must be reprocessed. Check for any errors reported in the log files and make any necessary corrections. Update the incompletely processed intervals to *state* = `NEW` to have the intervals reprocessed when *cron* executes *Archive* next.

Verify that the database rows have been successfully transferred to the archive database. First identify the rows on the operations database using the [*datatype*]-*read_data* query from the parameter file for the data type. Then, run the same query on the archive database. Compare the two records to ensure that the only differences are *dir*, *dfile*, and *foff*. An example for the parameterized query for reading continuous waveforms follows:

```
PRIARC-read_data="\
select distinct w.* from $(WFDISC) w, $(CHANGROUPS) cg \
where cg.sta=w.sta and cg.chan=w.chan \
  and w.time between %f and %f-.00001 and cg.name='%s' \
  and cg.class='%s' and cg.offdate=-1 \
order by w.sta, w.chan, w.time"
```

To inspect a single interval, extract the interval information from the database and construct a query using the *read_data* query as a model (replacing the parameters with the interval information). For example, the following interval is for the three-component station ULM:

```
INTVLID   CLASS   NAME  TIME           ENDTIME        STATE
--------  ------  ----  -------------  -------------- -----
12838944  PRIARC  ULM   992966400.000  992980800.000  DONE
```

Remove the parameterization from the *read_data* query to construct the following query:

```
select distinct w.* from idcx.wfdisc w, idcx.chan_groups cg
where cg.sta=w.sta and cg.chan=w.chan
  and w.time between 992966400.000 and 992980800.000-.00001
  and cg.name='ULM' and cg.class='PRIARC' and cg.offdate=-1
order by w.sta, w.chan, w.time;
```

Execute the query on the operations database. The following results are obtained:

```
STA CHAN TIME          WFID     CHANID  JDATE   ENDTIME
NSAMP SAMPRATE  CALIB     CALPER INSTYP S DA C
DIR                                 DFILE
FOFF      COMMID   LDDATE
--------------------------------------------------------------
ULM   be 992966400.000 57199441 110895  2001170 992980799.975
576000 40       .33500001 10     CMG3ES o s3 -
/data/amazon/cds/dl/ULM/ULMbe       020.w
0         -1       19-JUN-2001

ULM   bn 992966400.000 57199442 110896  2001170 992980799.975
576000 40       .33500001 10     CMG3ES o s3 -
/data/amazon/cds/dl/ULM/ULMbn       018.w
0         -1       19-JUN-2001

ULM   bz 992966400.000 57199443 110897  2001170 992980799.975
576000 40       .33500001 10     CMG3ES o s3 -
/data/amazon/cds/dl/ULM/ULMbz       044.w
0         -1       19-JUN-2001
```

Execute the query on the archive database. The **chan_groups** table must exist on the IDCX account in the archive database to use this query. The following results are obtained:

```
STA CHAN TIME          WFID     CHANID  JDATE   ENDTIME
NSAMP SAMPRATE  CALIB     CALPER INSTYP S DA C
DIR                                 DFILE
FOFF      COMMID   LDDATE
--------------------------------------------------------------
ULM   be 992966400.000 32023435 110895  2001170 992980799.975
576000 40       .33500001 10     CMG3ES o s3 -
/data/tiny/wave/idc/w/2001/170/     ArchULM.992966400.w
0         -1       22-JUN-2001
```

```
ULM   bn 992966400.000 32023436 110896  2001170 992980799.975
576000 40       .33500001 10     CMG3ES o s3 -
/data/tiny/wave/idc/w/2001/170/     ArchULM.992966400.w
1728000   -1       22-JUN-2001

ULM   bz 992966400.000 32023437 110897  2001170 992980799.975
576000 40       .33500001 10     CMG3ES o s3 -
/data/tiny/wave/idc/w/2001/170/     ArchULM.992966400.w
3456000   -1       22-JUN-2001
```

Compare the two results to ensure that the only differences between the data in the two accounts are the *wfid*, *dir*, *dfile*, and *foff* values.

Screening Log Files

Periodically screen the log files for *Archive* and *MSwriter* for any problems. Log files for *Archive* are written to `/logs/archiver/`. Log files for *MSwriter* are on the mass-storage server at `/local/lib/MSwriter/log/`. View the log file directory using the UNIX `ls -la` command. The following listing has been edited for length, but provides an example of each type of *Archive* log file:

```
% ls -la
-rw-r--r--   1 nmrd     nmrd 249 Jun 19 18:00 AUXWF_Archive.1
-rw-r--r--   1 nmrd     nmrd 249 Jun 19 18:00 AUXWF_Archive.2
-rw-r--r--   1 nmrd     nmrd 249 Jun 19 18:00 AUXWF_Archive.3
-rw-r--r--   1 nmrd     nmrd 432 Jun 19 03:04 AUXWF_Archive.4
-rw-r--r--   1 nmrd     nmrd 432 Jun 18 03:04 AUXWF_Archive.5
-rw-r--r--   1 nmrd     nmrd 928 Jun 21 05:03 MSG_Archive
-rw-r--r--   1 nmrd     nmrd 604 Jun 20 20:06 MSG_Archive.1
-rw-r--r--   1 nmrd     nmrd 928 Jun 20 05:03 MSG_Archive.2
-rw-r--r--   1 nmrd     nmrd 604 Jun 19 20:07 MSG_Archive.3
-rw-r--r--   1 nmrd     nmrd 928 Jun 19 05:03 MSG_Archive.4
-rw-r--r--   1 nmrd     nmrd 604 Jun 18 20:07 MSG_Archive.5
-rw-r--r--   1 nmrd     nmrd 736 Jun 21 04:17 PRIARC_Archive
-rw-r--r--   1 nmrd     nmrd 251 Jun 21 02:03 PRIARC_Archive.1
-rw-r--r--   1 nmrd     nmrd 736 Jun 21 00:18 PRIARC_Archive.2
-rw-r--r--   1 nmrd     nmrd 251 Jun 20 22:03 PRIARC_Archive.3
-rw-r--r--   1 nmrd     nmrd 736 Jun 20 20:15 PRIARC_Archive.4
-rw-r--r--   1 nmrd     nmrd 221 Jun 20 18:42 PRIARC_Archive.5
-rw-r--r--   1 nmrd     nmrd 430 Jun 21 04:00 STACAP_Archive
```

```
-rw-r--r--   1 nmrd     nmrd 511 Jun 20 19:00 STACAP_Archive.1
-rw-r--r--   1 nmrd     nmrd 430 Jun 20 04:00 STACAP_Archive.2
-rw-r--r--   1 nmrd     nmrd 251 Jun 19 19:00 STACAP_Archive.3
-rw-r--r--   1 nmrd     nmrd 430 Jun 19 04:00 STACAP_Archive.4
-rw-r--r--   1 nmrd     nmrd 251 Jun 18 19:00 STACAP_Archive.5
```

A corresponding log file is written when *Archive* is executed for each different archive data type. The example directory contents show four types of *Archive* logs (AUXWF, MSG, PRIARC, and STACAP).

The current file for each data type is the file name without a numerical suffix. The log file for the previous instance is *filename*`.1`. When the program is executed again, *filename*`.1` becomes *filename*`.2` and this renumbering cycle continues until log files recycle. Use the UNIX `grep` command to search for problems:

```
% grep fatal *_Archive.*

AUXWF_Archive.19:/cmss/rel/bin/Archive[fatal]:
Could not open database idcx/@machine, vendor oracle

MSG_Archive.20:/cmss/rel/bin/Archive[fatal]:
unable to archive data for intvlid=12692624.
Error Message: unable to write database records to Archive database

% grep error *_Archive.*

AUXWF_Archive.12:/cmss/rel/bin/Archive[error]:
unable to archive data for intvlid=12746574.
Error Message: SEND_DATA: Failed to open file
/aux/2001/158/idcaux56534133.w (No such file or directory)

AUXWF_Archive.13:/cmss/rel/bin/Archive[error]:
unable to archive data for intvlid=12746574.
Error Message: SEND_DATA: Failed to open file
/aux/2001/158/idcaux56534131.w (No such file or directory)
```

INTERPRETING ERROR MESSAGES

Common error messages written to log files, their description, and actions you can take to correct the problems are listed below.

Message:	`Failed to open file` */dir/file* `(No such file or directory)`
Description:	A waveform file could not be opened, usually because the file was missing. This error is fatal; the program exits.
Action:	Remove the **wfdisc** record that references the missing data.

Message:	`Could not open database` *database*`, vendor oracle`
Description:	*Archive* was unable to connect to the database. This error is fatal; the program exits.
Action:	Check that the database instance is running. Verify that the database name and password in the par file are correct.

Message:	`unable to get ARCH_DATA_TYPE row:` *datatype*
Description:	*Archive* was unable to obtain the **arch_data_type** row for the *datatype*. This error is fatal; the program exits.
Action:	Check that the database instance is running and that the **arch_data_type** row exists for the *datatype* specified.

Message:	`unable to generate or update intervals`
Description:	Intervals could not be created or updated in the **interval** table. This error is fatal; the program exits.
Action:	Check that the database instance is running and that the **interval** table exists in the archive database. Check log files for any other indications of the cause.

Message:	`unable to update interval table to status` *status* `for intvlid` *identifier*
Description:	The **interval** table could not be updated. This error is fatal; the program exits.
Action:	Check that the database instance is running and that the **interval** row for the *identifier* exists.
Message:	`Fatal error encountered while getting data`
Description:	The database query for the operations database references failed. This error is fatal; the program exits.
Action:	Check that the database instance is running and that the data for the interval actually exists. Check that the [*datatype*]-*read_data* database query is correct.
Message:	`unable to archive data for intvlid=`*identifier* `Error Message:` *text*
Description:	A problem was encountered while communicating with *MSwriter*.
Action:	Check the network connection to *MSwriter*, that the mass-storage device has available disk space, and that permissions are properly set on the mass-storage filesystem. Check that the data files exist and are readable on the operations filesystem and that the referencing database rows have the correct *foff* and *size*.
Message:	`Failed query:` *text*
Description:	A database query to create, read or queue intervals failed.
Action:	Check that the database instance is running. Verify that the database queries run using the *debug_queries* flag; then test the query manually in an SQL*Plus window.

SOLVING COMMON PROBLEMS

This section provides instructions for solving common problems that occur in the software. Common problems with the Archiving Subsystem include *MSwriter* connection problems and excessive file sizes to be archived.

MSwriter Connection Problems

Archive is occasionally unable to connect to *MSwriter*. When this occurs, the archive interval reports *state* = `FAILED`. The *MSwriter* log file should indicate the problem. This could be the result of a mass store failure or a hung *MSwriter* process. After correcting the problem, log onto the mass-storage server and use the UNIX `ps -ef | grep` command to check for any orphaned *MSwriter* processes. Use the UNIX `kill` command on any *MSwriter* processes found:

```
% ps -ef | grep MSwriter

cmss 19799 1 0 Jun 20 pts/8 1:41 MSwriter par=/cmss/config/
app_config/archive/MSwriter_sock.par

% kill 19799
```

Although the extra *MSwriter* processes increase the system load on the mass-storage server, this does not reduce system reliability. Even with greater than 20 runaway *MSwriter*s, the data are archived properly (though the system load may be extremely high).

Confirm that the network connection is working by pinging the mass-storage server (for example, `ping tiny`) or using the UNIX `tail -f` command on the *Archive* and *MSwriter* log files. Use an SQL command to update failed intervals status to *state* = `NEW`. This ensures that intervals will be reprocessed when *cron* executes *Archive* again.

```
Update idcx.interval state='NEW'
where moddate=sysdate and state='FAILED' and class=[archiving class]
```

Error Recovery

To recover from most archiving problems, reset the *state* in the **interval** table from `FAILED` to `NEW`. If the interval processing fails a second time, there is a persistent error. Scan log files for clues to the source of the problem and make the necessary corrections as indicated in "Interpreting Error Messages" on page 39.

REPORTING PROBLEMS

The following procedures are recommended for reporting problems with the application software:

1. Diagnose the problem as far as possible.
2. Record information regarding symptoms and conditions at the time of the software failure.
3. Retain copies of relevant sections of the application log files.
4. Contact the provider or maintainer of the software for problem resolution if local changes of the environment or configuration are not sufficient.

Chapter 4: Installation Procedures

This chapter provides instructions for installing the software and includes the following topics:

- Preparation
- Executable Files
- Configuration Data Files
- Database
- Initiating Operations
- Validating Installation

Chapter 4: Installation Procedures

PREPARATION

The first step in preparing to install the Archiving Subsystem is to define the host and server machines. The host (where *Archive* is run) is the computer with the data for the particular data type (for example, DAQ1HOST for continuous waveforms). The server (where *MSwriter* is run) is the mass-storage server. After the hardware is defined, define the data type-specific archiving destinations and add these site-specific paths to the parameter files (typically in `shared.par`). Details of the parameter file editing are provided in "Parameter File Organization" on page 46.

After the hardware mapping is determined, install the binary and parameter files. Install *Archive* in the directory `$(RELBIN)` of the host and install *MSwriter* on a local filesystem of the server. After binaries are installed, the subsystem must be configured. This involves adding entries to the `crontab` for the host and updating the `inetd.conf` and `services` files for the server. Details are provided later in this chapter.

Obtaining Released Software

Obtain the software via FTP from a remote site or via a physical medium such as tape or CD-ROM. The software and associated configuration data files are stored as one or more tar files. Transfer the software and data files to an appropriate location on a local hard disk. Untar the tar files into a standard UNIX directory structure.

Hardware Mapping

Select the hardware on which to run the software components. Software components are generally mapped to hardware to be roughly consistent with the software configuration model; however, with the Archiving Subsystem, the hardware mapping is driven by a desire to minimize network load. To accomplish that, *Archive* hosts should be the computers hosting the data being archived. For example, the archiving of primary stations (*datatype* = `PRIARC`) should reside on the disk loop host (DAQ1HOST). Similarly, Auxiliary Archiving should be on the machine with the auxiliary data, and so forth. The archiving destination (server) is the mass-storage server.

UNIX Filesystem

The Archiving Subsystem uses the UNIX filesystem to store executables, parameter files, and log files for the Archiving Subsystem.

EXECUTABLE FILES

Install *Archive*, *check_archive*, and *MSwriter* in `/cmss/rel/bin` and install *MSwriter* on a local partition of the mass-storage server.

CONFIGURATION DATA FILES

The parameter files provided with the software include default values for all parameters. Archiving Subsystem par files use global variables (or references) where possible; however, both *Archive* and *MSwriter* par files include paths that must be edited for the IDC environment. Install parameter files in `/cmss/config/app_config/archive`.

CAUTION: Typically, you are free to modify the parameter files; however, Archiving Subsystem parameter files contain critical SQL statements that control interval generation and the archiving flow. Changes to these queries can result in archiving failures, and modification is not recommended. Critical SQL statements are found below the following comment in the parameter file:

```
# parameters after this line are SQL statements that are required
# for the archive to function correctly
# These parameters are considered part of the release; updates
# to the parameters below this line should be handled via
# software modification requests
```

Parameter File Organization

The `Archive.par` file contains two sections of parameter settings. The first half of the parameter file contains configuration specifications including the location of the mass-storage device, database account, log file location, and so forth. The second half of the parameter file contains configuration rules for each data type. These rules are in the form of SQL*Plus statements. When *Archive* is run with a data type option, *Archive* uses the values assigned for that particular data type in `Archive.par`

Site-specific Information

Archiving Subsystem parameter files contain site-specific information that must be configured for the IDC environment. Parameters for each of these par files pertinent to the Archiving Subsystem are listed in the example par files that follow. Italicized portions of the settings may need modification for the IDC environment.

Archive.par

```
par=$(IMSPAR)
#
# Setup
#
use_MSWriter=1
log-directory=$(LOGDIR)/archiver
log-name=$(datatype)_Archive
log-files=20
log-drm
log-gdi
exit_on_error
archiver_key=x
```

```
archive_machine=$($(ARCHOST)).$(domain)
append_date_to_dir
#
# Database
#
ops-database=$(IDCXDB)
arch-database=$(IDCX-ARCHDB)
vendor=$(DATABASE_VENDOR)
account=idcx
#
# Tables
#
MSGDISC=$(account).msgdisc
WFDISC=$(account).wfdisc
DLFILE=$(account).dlfile
FILEPRODUCT=$(account).fileproduct
FPDESCRIPTION=$(account).fpdescription
INTERVAL=interval
CHANGROUPS=$(account).chan_groups
#
# Interval Table
#
interval_lastid=lastid
interval_keyname=intvlid
#
# Primary Archiver
#
PRIARC-class=PRIARC
PRIARC-archive_directory=$(ARCHIVE_PRIARC_DIR)
PRIARC-Archive_Port=3888
PRIARC-fileprefix=Arch
PRIARC-filepost='.w'
PRIARC-lastid_table='lastid_arcdb';
PRIARC-lastid_keyname=wfid
PRIARC-archive-days=3
PRIARC-lookback-days=10
# states
PRIARC-interval-queued=QUEUED
PRIARC-interval-new=NEW
PRIARC-interval-retry=RETRY
```

```
PRIARC-interval-done=DONE
#
# Auxiliary Archiver
AUXWF-class=AUXARC
AUXWF-name=AUXNET
AUXWF-archive_directory=$(ARCHIVE_AUXWF_DIR)
AUXWF-Archive_Port=3888
AUXWF-fileprefix=AUX
AUXWF-filepost='.w'
AUXWF-lastid_table='lastid_arcdb';
AUXWF-lastid_keyname=wfid
AUXWF-lookback-days=20
AUXWF-archive-days=7
AUXWF-interval-queued=QUEUED1
AUXWF-interval-new=NEW
AUXWF-interval-retry=RETRY
AUXWF-interval-done=DONE1
#
# stacap Archiver
STACAP-class=CAPARC
STACAP-name=STACAP
STACAP-prodtype="'STA_STATUS','CHAN_STATUS'"
STACAP-archive_directory=$(ARCHIVE_STACAP_DIR)
STACAP-Archive_Port=3888
STACAP-fileprefix=STACAP
STACAP-filepost='.fp'
STACAP-lookback-days=30
STACAP-archive-days=10
STACAP-interval-queued=QUEUED
STACAP-interval-new=NEW
STACAP-interval-retry=RETRY
STACAP-interval-arcqueued=ARCQUEUED
STACAP-interval-arcnew=ARCNEW
STACAP-interval-arcretry=ARCRETRY
#
# MSG Archiver
MSG-class=MSG
MSG-name=MSG
MSG-archive_directory=$(ARCHIVE_MSG_DIR)
MSG-Archive_Port=3888
```

```
MSG-fileprefix=MSG
MSG-filepost='.msg'
MSG-lookback-days=49
MSG-archive-days=10
MSG-intervals-per-day=12
MSG-interval-queued=QUEUED
MSG-interval-new=NEW
MSG-interval-retry=RETRY
MSG-interval-arcqueued=ARCQUEUED
MSG-interval-arcnew=ARCNEW
MSG-interval-arcretry=ARCRETRY
#
# FrameStore archiver
FSARC-class=FSARC
FSARC-prodtype=framestore
FSARC-archive_directory=$(ARCHIVE_FSARCH_DIR)
FSARC-Archive_Port=3888
FSARC-filepost='.fs'
FSARC-lookback-days=40
FSARC-interval-queued=QUEUED
FSARC-interval-new=NEW
FSARC-interval-retry=RETRY
FSARC-interval-arcqueued=ARCQUEUED
FSARC-interval-arcnew=ARCNEW
FSARC-interval-arcretry=ARCRETRY
#
# parameters after this line are SQL statements that are required
# for the archive to function correctly
#
# These parameters are considered par of the release; updates to
# the parameters below this line should be handles via software
# modification requests
#
interval-update="update $(INTERVAL) \
set state='%s', moddate=to_date('%s', 'YYYYMMDD HH24:MI:SS') \
where intvlid='%d'"
#
# waveforms: Primary
#
PRIARC-interval_create="select distinct -1 intvlid, \
    '$(PRIARC-class)' class, cg.name name, dl.time time, \
    dl.time+dl.tlen endtime, '$(PRIARC-interval-new)' state, \
    to_date('{current_lddate}','YYYYMMDD HH24:MI:SS') moddate, \
    to_date('{current_lddate}','YYYYMMDD HH24:MI:SS') lddate \
from $(DLFILE) dl, $(CHANGROUPS) cg \
where cg.sta=dl.sta and cg.chan=dl.chan \
  and dl.time > {current_epoch} - 86400*$(PRIARC-lookback-days) \
  and cg.class='$(PRIARC-class)' \
minus \
select distinct -1 intvlid, '$(PRIARC-class)' class, cg.name name,\
    i.time time, i.endtime, '$(PRIARC-interval-new)' state,\
    to_date('{current_lddate}','YYYYMMDD HH24:MI:SS') moddate,\
    to_date('{current_lddate}','YYYYMMDD HH24:MI:SS') lddate \
from $(CHANGROUPS) cg, $(INTERVAL) i
where cg.name=i.name and cg.class='$(PRIARC-class)' \
  and i.class='$(PRIARC-class)' \
  and i.time > {current_epoch} - 86400*$(PRIARC-lookback-days)"
#
PRIARC-interval_update="update $(INTERVAL) \
  set state='$(PRIARC-interval-queued)', \
  moddate=to_date('{current_lddate}','YYYYMMDD HH24:MI:SS') \
where class='$(PRIARC-class)' and state in \
    ('$(PRIARC-interval-new)','$(PRIARC-interval-retry)')\
  and time < {current_epoch} - $(PRIARC-archive-days)*86400 \
  and class='$(PRIARC-class)'"
#
PRIARC-interval_select="select * from $(INTERVAL) \
where state='$(PRIARC-interval-queued)' and class='$(PRIARC-class)'\
order by time"
#
PRIARC-read_data="select distinct w.* \
from $(WFDISC) w, $(CHANGROUPS) cg \
where cg.sta=w.sta and cg.chan=w.chan \
  and w.time between %f and %f-.00001 \
  and cg.name='%s' and cg.class='%s' and cg.offdate=-1 \
order by w.sta, w.chan, w.time";
#
# waveforms: Auxiliary
#
AUXWF-interval_create="select distinct -1 intvlid, \
    '$(AUXWF-class)' class, '$(AUXWF-name)' name, \
    (to_date(m.idate,'YYYYDDD')-to_date('1970/01/01 00:00:00',\
    'YYYY/MM/DD HH24:mi:ss'))*86400 time, \
    (to_date(m.idate,'YYYYDDD')+1-to_date('1970/01/01 00:00:00',\
    'YYYY/MM/DD HH24:mi:ss'))*86400 endtime, \
    '$(AUXWF-interval-new)' state, \
    to_date('{current_lddate}','YYYYMMDD HH24:MI:SS') moddate, \
    to_date('{current_lddate}','YYYYMMDD HH24:MI:SS') lddate \
from $(MSGDISC) m \
where m.idate > to_char(to_date('{current_lddate}',\
    'YYYYMMDD HH24:MI:SS')-$(AUXWF-lookback-days),'YYYYDDD') \
minus \
select distinct -1 intvlid, '$(AUXWF-class)' class, \
    '$(AUXWF-name)' name, i.time, i.endtime, \
    '$(AUXWF-interval-new)' state, \
    to_date('{current_lddate}','YYYYMMDD HH24:MI:SS') moddate, \
    to_date('{current_lddate}','YYYYMMDD HH24:MI:SS') lddate \
from $(INTERVAL) i \
where i.time > {current_epoch} -86400*$(AUXWF-lookback-days) \
  and i.class='$(AUXWF-class)' and i.name='$(AUXWF-name)'"
#
AUXWF-interval_update="update $(INTERVAL) \
set state='$(AUXWF-interval-queued)', \
    moddate=to_date('{current_lddate}','YYYYMMDD HH24:MI:SS') \
where class='$(AUXWF-class)' \
  and state in ('$(AUXWF-interval-new)', '$(AUXWF-interval-retry)')\
  and time < {current_epoch} - $(AUXWF-archive-days)*86400"
#
AUXWF-interval_select="select * from $(INTERVAL) \
where state='$(AUXWF-interval-queued)' and class='$(AUXWF-class)'\
order by time"
#
AUXWF-read_data="select distinct w.* \
from $(WFDISC) w, $(CHANGROUPS) cg \
where cg.sta=w.sta and cg.chan=w.chan \
  and w.time between %f and %f-.00001 \
  and cg.name='%s' and cg.class='%s' and cg.offdate=-1 \
order by w.sta, w.chan, w.time";
#
# file products: STACAP
```

```
#
STACAP-interval_create="select distinct -1 intvlid, \
    '$(STACAP-class)' class, '$(STACAP-name)' name, \
    floor (fp.time/86400)*86400 time, \
    floor((fp.time+86400)/86400) *86400 endtime, \
    '$(STACAP-interval-new)' state, \
    to_date('{current_lddate}','YYYYMMDD HH24:MI:SS') moddate, \
    to_date('{current_lddate}','YYYYMMDD HH24:MI:SS') lddate \
from $(FILEPRODUCT) fp \
where fp.typeid in
    (select typeid from $(FPDESCRIPTION) \
    where prodtype in ($(STACAP-prodtype)))
  and fp.time > {current_epoch} -$(STACAP-lookback-days)*86400 \
minus   \
select distinct -1 intvlid, '$(STACAP-class)' class, \
    '$(STACAP-name)' name, i.time, i.endtime, \
    '$(STACAP-interval-new)' state, \
    to_date('{current_lddate}','YYYYMMDD HH24:MI:SS') moddate, \
    to_date('{current_lddate}','YYYYMMDD HH24:MI:SS') lddate \
from $(INTERVAL) i \
where i.time > {current_epoch} -86400*$(STACAP-lookback-days) \
  and i.class='$(STACAP-class)' and i.name='$(STACAP-name)'"
#
STACAP-interval_update="update $(INTERVAL) \
set state='$(STACAP-interval-queued)', \
    moddate=to_date('{current_lddate}','YYYYMMDD HH24:MI:SS') \
where class='$(STACAP-class)' and state in \
    ('$(STACAP-interval-new)','$(STACAP-interval-retry)')\
  and time < {current_epoch} - $(STACAP-archive-days)*86400"
#
STACAP-interval_select="select * from $(INTERVAL) \
where state='$(STACAP-interval-queued)' \
  and class='$(STACAP-class)' \
order by time"
#
STACAP-read_data="select distinct f.* from $(FILEPRODUCT) f \
where f.typeid in \
    (select typeid from $(FPDESCRIPTION) \
      where prodtype in ($(STACAP-prodtype))) \
  and f.time between %f and %f-.00001 \
order by f.sta, f.time";
#
# Messages
#
MSG-interval_create="select distinct -1 intvlid, \
    '$(MSG-class)' class, '$(MSG-name)' name, \
    trunc(m.itime/(86400/$(MSG-intervals-per-day)),\
    0)*86400/$(MSG-intervals-per-day) time, \
    trunc(m.itime/(86400/$(MSG-intervals-per-day))+1,\
    0)*86400/$(MSG-intervals-per-day) endtime, \
    '$(MSG-interval-arcnew)' state, \
   to_date('{current_lddate}','YYYYMMDD HH24:MI:SS') moddate, \
   to_date('{current_lddate}','YYYYMMDD HH24:MI:SS') lddate \
from $(MSGDISC) m \
where m.idate > to_char(to_date('{current_lddate}',\
    'YYYYMMDD HH24:MI:SS')-$(MSG-lookback-days) , 'YYYYDDD') \
minus   \
select distinct -1 intvlid, '$(MSG-class)' class,\
    '$(MSG-name)' name, i.time, i.endtime, \
    '$(MSG-interval-arcnew)' state, \
    to_date('{current_lddate}','YYYYMMDD HH24:MI:SS') moddate, \
    to_date('{current_lddate}','YYYYMMDD HH24:MI:SS') lddate \
from $(INTERVAL) i \
where i.time > {current_epoch} -86400*$(MSG-lookback-days) \
  and i.class='$(MSG-class)' and i.name='$(MSG-name)'"
#
MSG-interval_update="update $(INTERVAL) \
set state='$(MSG-interval-arcqueued)', \
    moddate=to_date('{current_lddate}','YYYYMMDD HH24:MI:SS') \
where class='$(MSG-class)' and state in \
    ('$(MSG-interval-arcnew)', '$(MSG-interval-arcretry)') \
  and time < {current_epoch} - $(MSG-archive-days)*86400"
#
MSG-interval_select="select * from $(INTERVAL) \
where state='$(MSG-interval-arcqueued)' and class='$(MSG-class)'\
order by time"
#
MSG-read_data="select distinct m.* from $(MSGDISC) m \
where m.itime between %f and %f and m.filesize > 10 \
order by m.msgid";
```

```
#
# FrameStore
#
FSARC-interval_create="select distinct -1 intvlid, \
    '$(FSARC-class)' class, \
    fp.sta||'.'||translate(to_char(revision,9999),'t ','t') name,\
    fp.time time, fp.endtime endtime, \
    '$(FSARC-interval-new)' state,\
    to_date('{current_lddate}','YYYYMMDD HH24:MI:SS') moddate, \
    to_date('{current_lddate}','YYYYMMDD HH24:MI:SS') lddate \
from $(FILEPRODUCT) fp \
where fp.typeid in (select typeid from $(FPDESCRIPTION) \
    where prodtype='$(FSARC-prodtype)') \
minus   \
select distinct -1 intvlid, '$(FSARC-class)' class, name, \
    i.time, i.endtime, '$(FSARC-interval-new)' state, \
    to_date('{current_lddate}','YYYYMMDD HH24:MI:SS') moddate, \
    to_date('{current_lddate}','YYYYMMDD HH24:MI:SS') lddate \
from $(INTERVAL) i \
where i.class='$(FSARC-class)' \
  and i.time > {current_epoch} - $(FSARC-lookback-days)*86400"
#
FSARC-interval_update="update $(INTERVAL) \
set state='$(FSARC-interval-queued)', \
    moddate=to_date('{current_lddate}','YYYYMMDD HH24:MI:SS') \
where class='$(FSARC-class)' \
  and state in ('$(FSARC-interval-new)','$(FSARC-interval-retry)') "
#
FSARC-interval_select="select * from $(INTERVAL) \
where state='$(FSARC-interval-queued)' and class='$(FSARC-class)' \
order by time"
#
FSARC-read_data="select distinct fp.* \
from $(FILEPRODUCT) fp, $(INTERVAL) i \
where fp.time between %f and %f-.00001 \
  and i.name='%s' and i.class='%s' \
  and fp.typeid in (select typeid from $(FPDESCRIPTION) \
    where prodtype='$(FSARC-prodtype)') \
  and fp.sta in (substr(i.name,1,INSTR(i.name,'.')-1)) \
  and fp.revision in (substr(i.name,INSTR(i.name,'.')+1))"
```

MSwriter_sock.par

```
# error logging
log_level=6
logfile=/local/lib/MSwriter/log/MSwriter.log
messages_in_log_file=1000
total_log_files=25
be_able_to_create_new_dir
# ipc
Archiver_key=09*-aQ.w
socket_timeout_secs=1800
# input source ("socket" or "tape")
source=socket
```

Global Par Files

The global parameter files `process.par`, `shared.par`, and `global.shenv` contain global variables for all operational software. These files contain site-specific path names that must be modified for the IDC environment. These files are located in directory `/cmss/config/system_specs/`. Parameters that may need modification are italicized. Some parameter files have been abbreviated to show only Archiving Subsystem relevant parameters.

process.par

```
IDCXDB=account/password@machine
par=$(CMS_CONFIG)/system_specs/shared.par
```

shared.par

```
DATABASE_VENDOR=oracle
LOGDIR=/logs
#Archive mass-store paths
ARCHIVE_PRIARC_DIR=/data/tiny/wave/idc/w
ARCHIVE_AUXWF_DIR=/data/tiny/wave/idc/w
ARCHIVE_STACAP_DIR=/data/tiny/wave/text/fileproducts
ARCHIVE_MSG_DIR=/data/tiny/wave/text/msg
MSwriter_DIR=/local/lib/MSwriter
```

```
# ARCHOST = Archive Host (Host of MSwriter)
ARCHOST=PROCHOST6
```

global.shenv

```
CMS_SCRIPTS=/cmss/scripts
IMSPAR=/cmss/config/system_specs/process.par
LD_LIBRARY_PATH=/usr/dt/lib:/cmss/rel/lib:/home/oracle/lib:/cmss/cots/tuxedo/lib:/opt/
SUNWspro/lib:/opt/SUNWmotif/lib:/usr/openwin/lib:/cmss/contrib/lib:/cmss/local/lib
LOCALHOME=/cmss/local
MANPATH=/usr/man:/usr/openwin/man:/usr/dt/man:/cmss/rel/doc/man:/cmss/local/man
ORACLE_HOME=/home/oracle
PATH=/usr/bin:/usr/sbin:/usr/dt/bin:/cmss/scripts/bin:/cmss/rel/bin:/cmss/contrib/bin:/ho
me/oracle/bin:/cms/cots/tuxedo/bin:/cmss/local/bin:/opt/SUNWspro/bin:/opt/SUNWmotif
/bin:/usr/openwin/bin:/home/licensed/frame_5.5/bin:/opt/atria/bin:/opt/local/bin:/opt/loc
al/adobe/Acrobat3/bin:/etc:/usr/ccs/bin:/usr/ucb:.
```

DATABASE

This section describes database elements required for the operation of this software component including accounts, tables, and initialization of the **lastid** table.

Accounts

Determine the account that *Archive* should use. This account should be on the operations database and must have read permission for the **wfdisc**, **fileproduct**, and **msgdisc** tables in the IDCX account on the operations database. The account must have write permission for the **wfdisc**, **fileproduct**, and **msgdisc** tables on the archive database.

Tables

You may need to create and populate database tables in the archive database. Log onto the archive database and verify that the tables **interval**, **arch_data_type**, and **chan_groups** exist.

Creating Tables

Use the SQL commands in this section to create the **interval**, **arch_data_type**, and **chan_groups** tables if they are absent from the archive database.

To create the **interval** table use the following SQL commands:

```
create table INTERVAL (
intvlid      NUMBER(8)   NOT NULL,
class        VARCHAR2(16)NOT NULL,
name         VARCHAR2(20)NOT NULL,
time         FLOAT(53),
endtime      FLOAT(53),
state        VARCHAR2(16),
moddate      DATE,
lddate       DATE
) tablespace IDC
storage ( initial 50m next 10m pctincrease 0 ) pctfree 10;
create index INCLASSX on INTERVAL(CLASS,NAME,ENDTIME)
tablespace IDCNDX
storage ( initial 20m next 5m pctincrease 0 ) pctfree 0 ;
create index INENDX on INTERVAL(ENDTIME) tablespace IDCNDX
storage ( initial 20m next 5m pctincrease 0 ) pctfree 0 ;
create index INMODX on INTERVAL(MODDATE) tablespace IDCNDX
storage ( initial 20m next 5m pctincrease 0 ) pctfree 0 ;
grant SELECT on INTERVAL to PUBLIC with grant option;
grant UPDATE on INTERVAL to SEL1 ;
grant UPDATE on INTERVAL to SEL2 ;
grant UPDATE on INTERVAL to SEL3 ;
grant INSERT on INTERVAL to LEB ;
```

To create the **arch_data_type** table use the following SQL commands:

```
create table arch_data_type (
ARCHID                                  NUMBER(8),
DATATYPE                                VARCHAR2(24),
TABLE_NAME                              VARCHAR2(32),
PRIMARYKEY                              VARCHAR2(24),
SECONDKEY                               VARCHAR2(24),
MERGE_ADJACENT                          VARCHAR2(2),
NOTE_MISSING_DATA                       VARCHAR2(2),
ONDATE                                  NUMBER(8),
OFFDATE                                 NUMBER(8),
LDDATE                                  DATE
);
grant SELECT on ARCH_DATA_TYPE to PUBLIC;
```

To create the **chan_groups** table use the following SQL commands:

```
create table chan_groups (
CLASS                                   VARCHAR2(16),
NAME                                    VARCHAR2(16),
STA                                     VARCHAR2(6),
CHAN                                    VARCHAR2(8),
DURATION                                NUMBER,
INWFACTIVITY                            NUMBER(1),
ONDATE                                  NUMBER(8),
OFFDATE                                 NUMBER(8),
LDDATE                                  DATE
);
create index cgstachanx on chan_groups(sta, chan);
create index cgclassnamex on chan_groups(class, name);
grant SELECT on CHAN_GROUPS to PUBLIC;
```

Initializing Tables

After creating the tables and indexes, add rows to initialize the **arch_data_type** and **chan_groups** configuration tables. For example:

```
insert into arch_data_type
values ( 1, 'PRIARC','wfdisc','sta','time','y','n',-1,-1,sysdate);
insert into arch_data_type
values ( 2, 'AUXARC','wfdisc','sta','time','y','n',-1,-1,sysdate);

insert into chan_groups
select 'PRIARC', a.net, s.sta, s.chan, -1, -1, -1, -1, sysdate
from idcx.sitechan s, idcx.affiliation a, idcx.affiliation a2
where a2.net in ('PRI','INFRA', 'HYDRO')
  and s.chan not like '%b' and a2.sta=a.net and s.sta=a.sta;
```

The Archiving Subsystem uses an archive **lastid** table stored in the archive database. The **lastid** table functions as a counter by storing a numerical value for the latest identification number for each message. The attribute *wfid* is read and incremented by *Archive* each time a new record is added to the **wfdisc** table. This attribute must be present in the **lastid** table and must be initialized to a non-negative value (0 is acceptable). An example of these attributes in the **lastid** table in the archive account at the IDC follows:

```
SQL> desc LASTID_ARCDB;
Name                    Null?    Type
---------------------- -------- -------------
 KEYNAME                NOT NULL VARCHAR2(15)
 KEYVALUE               NOT NULL NUMBER(8)
 LDDATE                         DATE

SQL> select * from LASTID_ARCDB;
KEYNAME           KEYVALUE LDDATE
--------------- ---------- ------------------------
wfid                395375 22-JUN-01
```

INITIATING OPERATIONS

Creating crontab Entries

The Archiving Subsystem runs daily from *cron* and `crontab` entries must be created for each configured data type. Create the following four `crontab` entries on the machine where *Archive* runs:

```
#
#----------Run Archive for Auxiliary Waveforms
#
0 3,18 * * * ( env ‘cat $(GLOBALDIR)/env/global.shenv‘ run_archive
  AUXWF > /dev/null 2>&1 )

#
#----------Run Archive for Messages
#
0 5,20 * * * ( env ‘cat $(GLOBALDIR)/env/global.shenv‘ run_archive
MSG   > /dev/null 2>&1 )
```

```
#
#----------Run Archiver for Continuous Waveforms
#
  03 18,20,22,0,2,4  * * * ( env ‘cat
/cmss/config/system_specs/env/global.shenv‘ run_archive PRIARC
>/dev/null 2>&1 )

#
#----------Run Archive for STACAP
#
  0 4,19 * * * ( env ‘cat $(GLOBALDIR)/env/global.shenv‘ run_archive
STACAP > /dev/null 2>&1 )
```

The `crontab` entries all reference a script called *run_archive,* which is located in `/cmss/scripts/bin/`. An example of the *run_archive* script follows:

```
#!/bin/csh -f
# Usage:  run_Archive datatype
# where datatype = [CONTWF,AUXWF,STACAP,MSGS]
# checks to see if Archive is already running for the
# specified datatype, if not, Archive is started.
# The check examines only the current user on the
# current host!
#
if ( ‘ps -fu $USER | egrep "Archive datatype=$1 " | egrep -v egrep |
wc -l‘ < 1 ) then echo "Starting Archive for $1"
       ‘imspar RELBIN‘/Archive datatype=$1 par=‘imspar
ARCHIVEDIR‘/Archive.par &
else
   echo "Archive for $1 already running..."
endif
```

Modifying inetd.conf and services

MSwriter requires that the following lines be added to the `inetd.conf` and `services` files of the archiving destination machine. These files are maintained by UNIX system administrators. The service and site-specific path in `inetd.conf` must be updated for the IDC environment, and the port numbers in `services` must match the values given to the variables [*datatype*]-*Archive_Port*. For example, if *PRIARC-Archive_Port* = `3888` and *AUXARC-Archive_Port* = `3334`, then the following lines are added to `inetd.conf`:

```
mass-store stream tcp nowait cmss
   /local/lib/MSwriter/bin/run_MSwriter run_MSwriter
mass-store-aux stream tcp nowait cmss
   /local/lib/MSwriter/bin/run_MSwriter run_MSwriter
```

In the `services` file, the following lines are added to match the service and port number:

```
mass-store      3888/tcp
mass-store-aux  3334/tcp
```

After you have created the `crontab` entries, installed the *run_archive* script, and the `inetd.conf` and `services` files of the mass-storage server have been appended, the Archiving Subsystem is ready for operation. The *cron* daemon invokes *Archive* automatically at the time configured for each data type in the `crontab` files.

VALIDATING INSTALLATION

Use the UNIX `ping` command on the mass-storage server and also use the `tail -f` UNIX command on the *MSwriter* log file to verify the network connection to the mass store. When *Archive* has run, view the archiving status of the intervals using the *Archive WorkFlow* GUI. Verify that all data types are being archived, including continuous waveforms, auxiliary waveforms, messages, and file products.

Use the program *check_archive* to verify the Archiving Subsystem installation. *check_archive* only verifies primary waveform archiving. *check_archive* compares data on the disk loops (operations filesystem) to data on the mass store (archive filesystem) on a sample-for-sample basis. *check_archive* also compares the **wfdisc** records and determines if they are identical. If all of the waveforms and **wfdisc** records are identical, *check_archive* updates the *state* of the corresponding **interval** record to `VERIFIED`; otherwise it updates the *state* to `NOTVERIF`.

You can run *check_archive* from the command line using a par file argument. The file `check_archive.par` should be located in directory `/cmss/config/app_config/archive/`. An example par file follows:

```
log-directory=log
log-name=checkarch.log
log-files=100
ops-database=operations_database
arch-database=archive_database
ops-account=wfdisc_account
class=PRIARC
vendor=oracle
```

The *operations_database* must contain the **interval** table.

To run *check_archive,* which verifies all intervals of *class* = `PRIARC` for which *state* = `DONE`, use the following command:

```
/cmss/rel/bin/check_archive \
par=/cmss/config/app_config/archive/check_archive.par
```

Use *WorkFlow* or SQL*Plus queries to check for any `NOTVERIF` records after running *check_archive.* The following SQL*Plus example makes this check:

```
select state, count(*) from interval where class='PRIARC'
group by state;
```

Use the following procedures to manually check data that could not be verified by running *check_archive* (in other words, for which **interval**.*state* = `NOTVERIF`).

1. Examine log files for the following errors:

   ```
   Unable to read wfdiscs from archive database
   ```

2. Check the **wfdisc**.*lddate* in the operations database for the offending **wfdisc** entry; if it is later than the archive time, then the data arrived after archiving and no further action is required.

3. Check log files for errors reporting that samples are different. Typically, any discrepancies are caused by clock drift, which results in the offset being miscomputed. For example:

   ```
   check_archive[warning]: intvlid 699554:Station PD04.sz has off-
   set of 238919 samples, time=957424776.000000, end-
   time=957425075.950000
   check_archive[info]: intvlid 699554:Station PD04.sz has 2270
   bytes different, time=957424776.000000, endtime=957425075.950000
   ```

   In this case, round the sample offset (reported in the log file) from 238919 to 238920 (the nearest multiple of 10), and manually verify using octal dump (*od*). Refer to the man page on *od* for more information on its use and its flags. The *od* verifications for the above warning are:

   ```
   od -c -N 18000 -j 1400400 \
       /data/spinach/cds/dl/PDAR/PD04sz/007.w >ops
   od -c -N 18000 -j 716760 \
       /data/tiny/wave/w/2000/125/archPDAR.957412800.w >arch
   ```

   Use the UNIX `diff` command to compare the two files. If there are no results written to `stdout`, the files are identical. For example:

   ```
   % diff ops arch
   ```

Refer to "Monitoring" on page 32 for more information on verifying Archiving Subsystem operation.

References

The following sources supplement or are referenced in this document:

[Gan79]	Gane, C., and Sarson, T., *Structured Systems Analysis: Tools and Techniques*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1979.
[IDC5.1.1Rev2]	Science Applications International Corporation, Veridian Pacific-Sierra Research, *Database Schema, (Part 1, Part 2, and Part 3)*, *Revision 2*, SAIC-00/3057, PSR-00/TN2830, 2000.
[IDC6.5.18]	Science Applications International Corporation, *Continuous Data Subsystem CD-1.1 Software User Manual*, SAIC-01/3006, 2001.
[IDC7.4.4]	Science Applications International Corporation, *Subscription Subsystem*, SAIC-98/3001, 1998.
[IDC7.5.1]	Science Applications International Corporation, *Archiving Subsystem,* SAIC-01/3013, 2001.

Glossary

Symbols

3-C

Three-component.

A

architecture

Organizational structure of a system or component.

architectural design

Collection of hardware and software components and their interfaces to establish the framework for the development of a computer system.

archive

Single file formed from multiple independent files for storage and backup purposes. Often compressed and encrypted.

ASCII

American Standard Code for Information Interchange. Standard, unformatted 256-character set of letters and numbers.

C

CD-ROM

Compact Disk–Read Only Memory.

child process

UNIX process created by the *fork* routine. The child process is a snapshot of the parent at the time it called *fork*.

client

Software module that gathers and presents data to an application; it generates requests for services and receives replies. This term can also be used to indicate the requesting role that a software module assumes by either a client or server process.

command

Expression that can be input to a computer system to initiate an action or affect the execution of a computer program.

computer software component

Functionally or logically distinct part of a computer software configuration item; possibly an aggregate of two or more software units.

computer software configuration item

Aggregation of software that is designated for configuration management and treated as a single entity in the configuration management process.

COTS

Commercial-off-the-shelf; terminology that designates products such as hardware or software that can be acquired from existing inventory and used without modification.

CSC

See computer software component.

CSCI

See computer software configuration item.

D

DACS

See Distributed Application Control System.

disk loop

Storage device that continuously stores new waveform data while simultaneously deleting the oldest data on the device.

Distributed Application Control System

This software supports inter-application message passing and process management.

F

fileproduct

(1) Object method for creating a database reference for any filesystem object. (2) Database table whose records describe files containing products.

filesystem

Named structure containing files in subdirectories. For example, UNIX can support many filesystems; each has a unique name and can be attached (or mounted) anywhere in the existing file structure.

fork

UNIX system routine that is used by a parent process to create a child process.

Frame Store

Disk directories and files used to store raw CD-1.1 protocol frames. A Frame Store is made up of frame sets.

FTP

File Transfer Protocol; protocol for transferring files between computers.

G

GB

Gigabyte. A measure of computer memory or disk space that is equal to 1,024 megabytes.

GUI

Graphical User Interface.

H

hr

Hour(s).

I

IDC

International Data Centre.

IMS

International Monitoring System.

instance

Running computer program. An individual program may have multiple instances on one or more host computers.

M

mass-storage device (mass store)

Physical device that is capable of storing exceptional data volumes as part of the filesystem. Typically, this is a tape or disk jukebox that uses media such as Compact Disks (CD), Digital Versatile Disks (DVD), and Digital Linear Tape (DLT).

MB

Megabyte. 1,024 kilobytes.

min

Minute.

N

network

Spatially distributed collection of seismic, hydroacoustic, or infrasonic stations for which the station spacing is much larger than a wavelength.

O

ORACLE

Vendor of the database management system used at the PIDC and IDC.

P

par

See parameter.

parameter

User-specified token that controls some aspect of an application (for example, database name, threshold value). Most parameters are specified using [*token* = *value*] strings, for example, `dbname=mydata/base@oracle`.

parameter (par) file

ASCII file containing values for parameters of a program. Par files are used to replace command line arguments. The files are formatted as a list of `[`*token* = *value*`]` strings.

S

SAIC

Science Applications International Corporation.

script

Small executable program, written with UNIX and other related commands, that does not need to be compiled.

server

Software module that accepts requests from clients and other servers and returns replies.

software unit

Discrete set of software statements that implements a function; usually a subcomponent of a CSC.

Solaris

Name of the operating system used on Sun Microsystems hardware.

SQL

Structured Query Language; a language for manipulating data in a relational database.

States Parties

Treaty user group who will operate their own or cooperative facilities, which may be National Data Centres.

T

tar

Tape archive. UNIX command for storing or retrieving files and directories. Also used to describe the file or tape that contains the archived information.

time series

Time ordered sequence of data samples. Typically a waveform or derived from waveforms, such as a beam.

Tuxedo

Transactions for UNIX Extended for Distributed Operations.

U

UNIX

Trade name of the operating system used by the Sun workstations.

W

waveform

Time-domain signal data from a sensor (the voltage output) where the voltage has been converted to a digital count (which is monotonic with the amplitude of the stimulus to which the sensor responds).

Index